INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 10, NO. 3, SEPTEMBER 2005, 210-217

Synthesis of an Event Based Supervisor for Deadlock Avoidance in Semiconductor Manufacturing Systems

Wenle ZHANG and Ziqiang John MAO

Abstract—**With the emerging of highly automated and flexible manufacturing systems in semiconductor fabrication, reliability and productivity of such systems require intelligent and complex control systems. Deadlock issue arises easily in these systems due to shared equipment usage and high production flexibility. This paper presents an event-based deadlock avoidance super visor that is able to efficiently and smartly avoid the deadlock state space explosion problem. The method is built upon a directed graph model of process flows. Concepts such as compound events, generalized strings, operating strings and deadlock strings are introduced in the paper. Major features in the proposed method include: i) it enables optimal deadlock free operation of regular systems; and ii) it runs in polynomial time (fast online computation) provided that the set of deadlock strings is calculated offline. Examples are provided to show the effectiveness of the method.**

Index terms—**manufacturing system, discrete event system, deadlock avoidance, digraph, circuit**

1. INTRODUCTION

Currently productivity and equipment utilization are critical to reduce the cost of products and to support the decreasing trend of average sale price (ASP) in semiconductor industry. The goal of manufacturing scheduling and deadlock avoidance is to ensure an optimal solution for the manufacturing operations, which maximizes product output, equipment utilization and availability; and minimizes the wafer process throughput-time and cost. Such solutions can save as much as millions of dollars per year.

A manufacturing operation is a process involved with equipment, wafers, and technicians. At each step, if any one of them is not available, a wafer processing step cannot continue, such as, equipment waits for wafers or wafers wait for equipment. For example, the multi-million dollar lithographic equipment is the key bottleneck component in a fab. The wafers to be processed in a photolithography step come from different process functional areas such as diffusion, thin film, polish etc. The optimized scheduling of wafer movements and litho equipment utilization are critical to the total throughput. Also, the wafers from different areas coming into one area can encounter traffic jam and slow down the process flow.

Manuscript received Dec 29, 2004; revised March 18, 2005. This paper is extended from "Synthesis of an Event Based Supervisor for Deadlock Avoidance in Semiconductor Manufacturing Systems" published in the Proc. of 2004 American Control Conference, Boston, USA.

Wenle Zhang is with School of Electrical Eng. & Computer Sci., Ohio University, Athens, OH 45701, USA (Email: zhangw@ohio.edu). Ziqiang John Mao is with Intel Corporation, Technology Manufacturing Engineering, 2200 Mission College Blvd., Santa Clara, CA 95052.

One way to minimize the throughput time is to increase the quantity of capital. After increasing some expensive equipment, more equipment idle time is observed. A fab (fabrication site) is always equipped with a certain number of similar equipment. For example, a fab may have 10-20 lithography steppers/scanners. Hence, the minimization of throughput time, equipment idle time and cost requires balanced optimization on both. Also, with the minimized number of machines, a deadlock-free wafer flow is required. A deadlock free system is crucial for manufacturing to achieve the minimal throughput time, minimal equipment idle time and maximum productivity.

Zhou and Jeng [17] presented a Petri net method for modeling, analysis and control of a general semiconductor manufacturing system. Modeling and performance analysis of cluster tools were given by Srinivasan [8]. Deadlock free scheduling of a track system for semiconductor fabrication was proposed by Yoon [13].

Deadlock detection, prevention and avoidance for general flexible manufacturing systems have been studied extensively. Some of the significant works [1, 2, 4, 9, 11, 16] have adopted Petri net (PN) models as a formalism to describe the manufacturing system. Banaszak and Krogh [1] proposed a *deadlock avoidance algorithm* (DAA) that developed a restriction policy to guarantee that no circular wait situations occur. Viswanadham *et al.* [9] developed a deadlock avoidance algorithm which suggested using a recovery mechanism in case of system deadlock. Zhou and DiCesare [16] developed the *sequential mutual exclusions* (SME) and *parallel mutual exclusions* (PME) concepts and derived the sufficient conditions for a PN containing such structures to be bounded, live, and reversible. Structural properties of PNs such as siphons and traps were used to determine potential deadlock situations in [2, 4].

Another formalism is to describe a manufacturing system using graphs [3, 5, 7, 10, 14, 15]. In this approach the vertices represent resources and arcs (edges) represent product part flows between resources. Cho *et al.* [3] developed the concept of bounded circuits with empty and non-empty shared resources to detect deadlock. Judd and Faiz [7] derived a set of static linear inequalities whose satisfaction avoided deadlock. Zhang *et al.* [14] quantified both necessary and sufficient conditions for deadlock to occur in a manufacturing system. Yalcin and Boucher [12] studied deadlock avoidance in flexible manufacturing systems using finite automata which are applicable to small systems only since the size of state space is exponential to the problem size.

However, the deadlock problem has not been well studied from the event point of view. This paper presents an event-based framework for deadlock avoidance in manufacturing systems. The major contribution of this paper is to propose a highly efficient event-based deadlock avoidance supervisor. The supervisor smartly avoids the deadlock state space explosion problem. The method is built upon a directed graph model of process flows. To derive results, we introduced compound events, operating strings, deadlock strings and other related concepts. There are two major benefits in our method. First, the method provides the optimal deadlock free operation that allows all live states for a large class of systems. The large class of systems is called regular systems which do not contain key resources, as discussed by [11]. Second, it runs in polynomial time, which means faster online computation, once the set of deadlock strings are computed offline. Examples and discussions are provided to show the effectiveness of the method.

2. The System Model and Deadlock Concept

A semiconductor manufacturing system consists of a finite number of equipments or resources, denoted as a set R, that include chambers, robots, buffers, etc.. There are a finite number of process types, denoted as a set P, to which jobs should follow. Each type $p \in P$ is described as a finite number of steps of operations that need to be performed on jobs of the type. We assume that each step is performed by exactly one resource. Thus a process p can be represented as a sequence of resources $p=r_1r_2\ldots r_m$, where m is the number of steps. Each resource $r \in R$ has a *capacity*, denoted as C_r (>1 for regular systems). The capacity of the system is $C_R = \Sigma\, C_r$. The condition $C_r > 1$ can be easily met since a resource may have a buffer, its processing capacity being greater than one, and/or stands for multiple identical units

For the purpose of deadlock avoidance, these systems are modeled by a directed graph, denoted as G, which is constructed from all process types. The graph $G = (R, A)$ consists of a set R of *vertices* and a set A of *directed arcs*. Each vertex represents a resource. A directed arc a is drawn from vertex r_1 to vertex r_2, if r_2 immediately follows r_1 in at least one process plan, denoted as $a= r_1r_2$. A *subgraph* $G_1 = (R_1, A_1)$ of G consists of a subset of the vertices and a subset of arcs of G such that all the arcs in A_1 connect vertices in R_1. From graph theory, we know that a *path* is defined as a sequence of vertices $r_0r_1r_2\ldots r_k$, and a *circuit* is a path with $r_0 = r_k$. A circuit c is a subgraph, $c = (R_c, A_c)$. A circuit is called *simple* if it does not contain any other circuit. Vertex r_1 is *reachable* from r_2 if there is a path from r_2 to r_1. A subgraph G_1 is a *strongly connected component* (SCC) if all vertices of R_1 are reachable from each other. When the system graph consists of more than one SCC, we can trim off all arcs that are not part of any SCC. So, G=trim(G) consists of all isolated SCCs.

2.1. Conceptual System Deadlock

Conceptual system deadlocks are formulated based on a cluster of tool configuration with a double blades center robot as shown in Fig. 1. Let us consider simplified process flows with three assumed job types:

Type A wafer: CH1 – Robot – CH2

Type B wafer: CH2 – Robot – CH3

Type C wafer: CH3 – Robot – CH1

where the notation "CH – Robot" means from Chamber to Robot.

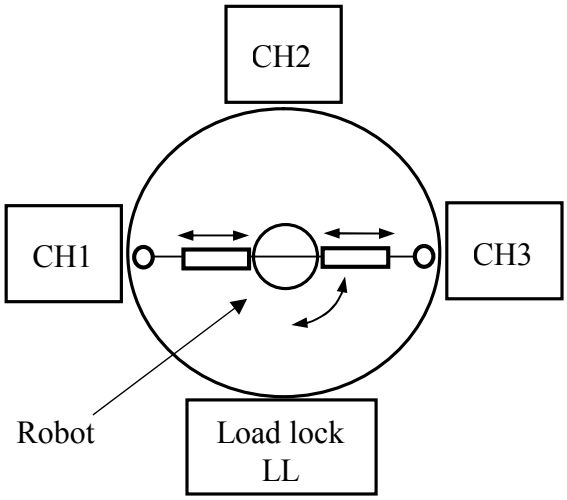

Fig. 1. Cluster tool configuration

Let r_1~r_3 to represent the Chambers and r_4 for Robot. Then,

$p_A = r_1–r_4–r_2$

$p_B = r_2–r_4–r_3$

$p_C = r_3–r_4–r_1$

Assume that the system is fully automated. Chambers r_1, r_2 and r_3 all have same capacity 3. Robot r_4 has capacity 2. And a finished product leaves the system automatically. The system graph is constructed in Fig. 2. The graph itself is a SCC.

Then, we can observe 6 direct deadlocks (without considering jobs in resources other than indicated):

i) when r_1 has 3 A-wafers and r_4 has 2 C-wafers

ii) when r_2 has 3 B-wafers and r_4 has 2 A-wafers

iii) when r_3 has 3 C-wafers and r_4 has 2 B-wafers

iv) when r_1 has 3 A-wafers, r_2 has 3 B-wafers and r_4 has 1 A and 1 C

v) when r_2 has 3 B-wafers, r_3 has 3 C-wafers and r_4 has 1 B and 1 A

vi) when r_3 has 3 C-wafers, r_1 has 3 A-wafers and r_4 has 1 C and 1 B

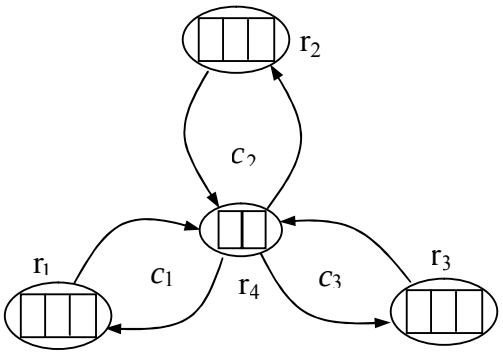

Fig. 2. Simplified directed graph

Observe that each deadlock is related to a circuit that is the root cause of a circular wait situation. For instance, the first deadlock corresponds to $c_1 = r_1\text{-}r_4\text{-}r_1$, a path from resource r_1 to r_4 to r_1; the fourth one relates to the union circuit $c_1 \cup c_2 = r_1\text{-}r_4\text{-}r_2\text{-}r_4\text{-}r_1$.

Formally, an (*immediate*) *deadlock* is defined as a circular wait situation [1] – a system state in which there exists a group of jobs and every job requests a resource that is held by another job in the group. An *impending deadlock* is a system state not in immediate deadlock, but will evolve into an immediate deadlock after a finite number of events.

2.2. System Event Model

Once a system is in operation, it changes its state with the occurrence of system events. In this paper, we consider only the following three types of high level events since they are most related to deadlock occurrences.

i) loading a job into the system,

ii) transporting a job from one step to next step, or

iii) unloading a finished job from the system.

Each process type p is associated with a sequence of events, $E_P = e_1e_2\ldots e_{Lp}$, corresponding to each process step including the loading/unloading steps, where L_P is the length of the job type, $L_p = m + 1$.

Definition 2.1: Each event e has a *source-step* and a *destination-step*, define an ss(ds) operation on an event e as a mapping: ss(e) (ds(e)) $\rightarrow$ N, which are the step numbers in the process type except that the 0 source-step and destination-step are for the load lock or warehouse for both raw and finished jobs.

For example, in Fig. 2, if we name events correspondingly to the three types of jobs with A, B and C, then A-job has event sequence $A_0A_1A_2A_3$, where A_0 corresponds to the event that loads an A-job into r_1, A_1 for the event that Robot r_4 picks up the job, A_2 for the event that r_4 loads r_2 with the job, and A_3 for the event that the job is unloaded from the system. Similarly, types B and C have event sequences $B_0B_1B_2B_3$ and $C_0C_1C_2C_3$, respectively. And ss(A_0) = 0, ds(A_0)=1, ds(B_2) = 3, ds(C_3) = 0 , etc.

An event is said to be *active* if an associated job exists in the system. An active event is *enabled* if the resource pointed to by the event's destination-step is free. An event with a source-step equal to 0 is always active and an active event with a destination-step equal to 0 is always enabled. A_0, B_0 and C_0 are always active, they are also enabled if the system is empty. If an A-job is loaded into r_1, then event A_1 is active. If there are three A-jobs in r_1, then A_0 is disabled.

Obviously, events with the same type ($E_P = e_1\ldots e_ie_{i+1}\ldots e_{Lp}$) have to occur in the order described by the process type. That is, e_i has to occur first in order to have an active event e_{i+1}. We say that e_i is the *causing event* of e_{i+1} and e_{i+1} is the *resulting event* of e_i, denoted by e_i+1, so ds(e_i) = ss(e_{i+1}) or ds(e_i) = ss(e_i+1).

For example, A_0 has to occur first in order to have an active A_1 event. Then A_0 is the *causing event* of A_1 and A_1 is the *resulting event* of A_0. Note, an event with source-step being 0 has no causing event and an event with destination-step being 0 has no resulting event.

3. Deadlock Avoidance Supervisor and Properties

From Section 2.1, we observe each deadlock corresponds to a circuit in a SCC. To synthesize our deadlock avoidance supervisor, we need to find all circuits in every isolated SCC of the system graph. Given a system graph, there are existing methods that find all circuits [6, 10]. In the following, we assume that all circuits are given.

Notice that many resources in the system are shared among multiple types of jobs. So, each resource is associated with a set of shared events.

Definition 3.1: The *set of shared events* of resource r consists of events whose destination-step matches the resource or the source-step matches it. This set can be further divided into two subsets: the set of events e entering resource r (ds(e) $\rightarrow$ r) – called *entering event set*, denoted by $E_{en}(r)$; and the set of events e leaving the resource (ss(e) $\rightarrow$ r) – called *exiting event set*, denoted by $E_{ex}(r)$.

For example, resource r_1 has a shared event set: {A_0 A_1 C_2 C_3}, where A_0 and C_2 are events corresponding to jobs entering r_1, *i.e.*, $E_{en}(r_1)$ = {A_0 C_2}, and A_1 and C_3 are events corresponding to jobs leaving r_1, *.e.*, $E_{ex}(r_1)$ ={A_1 C_3}. After A_0 of $E_{en}(r_1)$ occurs, the underlying job is in resource r_1 and after A_1 of $E_{ex}(r_1)$ occurs, the same job is out of r_1. These sets of events can be obtained by analyzing all the process types.

Definition 3.2: A *compound event* is defined on a resource as a number of occurrences (less than the capacity of the resource) followed by a subset of the shared event set of the resource.

For example, for r_1, we have 2(A_0) or 2(A_1,C_3), where an active compound event 2(A_1,C_3) can mean 2 active A_1 events, or 2 active C_3 events, or 1 active A_1 event and 1 active C_3 event.

Definition 3.3: A sequence of events is called a *string*. A string is *generalized* if it contains compound events.

A normal event e can be considered as a compound event in the form of $1(e)$. Then, both normal events and compound events are simply referred to as events, and both normal and generalized strings are simply referred to as strings.

As mentioned earlier, deadlock is related to circuits. Entering/exiting event sets can be naturally extended to a set of resources and thus a circuit.

The *entering event set of circuit* $c = (R_c, A_c)$ consists of all events e with ds(e) pointing to a resource of R_c and ds(e+1) also pointing to a resource of R_c, denoted by $E_{en}(c)$. The *exiting event set of circuit* $c = (R_c, A_c)$ consists of all events e with ss(e) pointing to a resource of R_c and ds(e) pointing to a resource not in R_c, denoted by $E_{ex}(c)$. For example, in Fig. 2, c_1 has $E_{en}(c_1) = \{A_0, C_1\}$ and $E_{ex}(c_1) = \{A_2, C_3\}$. Also, each circuit is assigned an operating string and a deadlock string.

Definition 3.4: An *operating string* of a circuit is the sequence of events that have occurred so far on the circuit, denoted as s_o.

For example, the operating string for c_1 after some time is $s_o = 2(A_0)\text{-}A_1\text{-}A_2$. What s_o says is that so far A_0 has occurred twice, A_1 and A_2 have occurred once. The result is that one A-job at r_1 and one A-job at r_3.

Definition 3.5: A *deadlock string* of a circuit c is defined as a string of events that have occurred on the circuit, which generates no active event in $E_{ex}(c)$ and no enabled event exists on the circuit, denoted as s_d.

For a circuit to be in deadlock, it has to be filled with jobs up to capacity and generates a set of active events that are not enabled, which means that no jobs can leave the circuit and both source resource and destination resource of all active events are on the circuit.

Example 3.1: In Fig. 2, when circuit c_1 is in deadlock (the first deadlock), as shown in Fig. 3, it has active event $3(A_1)$ and active event $2(C_2)$. Then the set that needs to be found is $\{3(A_1), 2(C_2)\}$. The set can then be translated into a *deadlock string* by finding the corresponding causing events and formulating into a string, the causing event of A_1 is A_0 and the causing event of C_2 is C_1, so the deadlock string in this case is $s_{d1} = 3(A_0)\text{-}2(C_1)$.

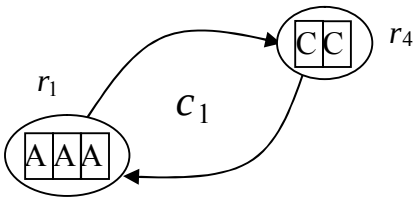

Figure 3. Deadlock on circuit c_1

Algorithm 3.1: Find the deadlock string for a circuit c = (R_c, A_c)

Input: P – set of process types, E_P – event set for each process type,

Output: s_d – deadlock string of c

```
First find E_ex(c)
For each r in R_c
E_r = {} – set of event sharing resource r
For each process type p in P
  For each event e in E_P
    Let r_e = p(ss(e)) be the resource at step ss(e)
    If r == r_e and e ∉ E_ex(c)
       Add e to E_r
  End for
End for
Append compound event C_r(E_r) to s_d
End for
S_d = translate each event of s_d into causing event
```

To avoid deadlock, we need to analyze events related to each circuit and simplify the operating string so that it can be used to check against the deadlock string of the circuit. The type of simplification we are interested in is to remove any causing events of the recently occurred event from the operating string. That is, the operating string is updated by filtering each new event according to the projection operation defined below.

First, the *ss* (*ds*) operation for an event can be easily extended to a string s as a mapping: ss(s) (ds(s)) $\rightarrow N^{|s|}$, that is, to give the set of source (destination)-step of all events in the string. For example, if $s_o = 2(A_0)\text{-}A_1\text{-}A_2$, then ss($s_o$) = {0, 0, 1, 2}.

Projection Operation: Given a circuit $c = (R_c, A_c)$, an event e such that ds(e) pointing to a resource of R_c or $e \in E_{ex}(c)$, define a projection operation on the operating string s_o of c such that,

i) If ss(e) $\in$ ds(s_o) and e_1 is the first event in s_o, such that ss(e) = ds(e_1), then e_1 is removed from s_o, that is, $s_o = s_o - e_1$; and

ii) If $e \notin E_{ex}(c)$, then $s_o = s_o + e$.

This operation is denoted as, $s_o = P(s_o, e)$.

Here, operation i) means to remove the first causing event of e from s_o and ii) means that if event e exits the circuit, then

it should not stay in the operating string. For example, in Fig. 2, given the operating string for circuit c_1 is $s_{o1} = A_0\text{-}A_1$, meaning one A-job at r_1 and one A-job at r_4, (i) if event A_0 occurs, then since it has no causing event and $A_0 \notin E_{ex}(c_1) = \{A_2, C_3\}$, so $s_{o1} = P(s_{o1}, A_0) = 2A_0\text{-}A_1$; (ii) if A_1 occurs, then since its causing event is A_0 and $A_1 \notin E_{ex}(c_1)$, so $s_{o1} = P(s_{o1}, A_1) = 2A_1$; and (iii) if A_2 occurs, since its causing event is A_1 and $A_2 \in E_{ex}(c_1)$, then $s_{o1} = P(s_{o1}, A_1) = A_0$.

Algorithm 3.2: String matching algorithm

Input: s_d – deadlock string of a circuit with length $|s_d|$

s_o – operating string of the circuit with length $|s_o|$

Output: true for a match and false for a non-match

```
If |s_o|<|s_d|, then return false
While s_o is not empty do
  Remove first event e from s_o
  If s_d contains e, then
    remove one occurrence of e from s_d
end while
if s_d is empty, then return true
else return false
```

Example 3.2: In Fig. 2, circuit $c_5 = c_2 \cup c_3$ has deadlock (the fifth deadlock, Fig. 4) string $s_{d5} = 3(B_0)\text{-}2(A_1,B_1)\text{-}3(C_0)$. If its operating string $s_{o5} = B_1\text{-}3B_0\text{-}3C_0$, then since $|s_{o5}| = 7 < |s_{d5}| = 8$, Algorithm 3.2 returns false immediately. But if $s_{o5} = A_1\text{-}B_1\text{-}3B_0\text{-}3C_0$, then

i) When the first time looping through Algorithm 3.2, we have $e = A_1$, $s_{o5} = B_1\text{-}3B_0\text{-}3C_0$ and $s_{d5} = 3(B_0)\text{-}1(A_1,B_1)\text{-}3(C_0)$.

ii) The second time looping through Algorithm 3.2, we have $e = B_1$, $s_{o5} = 3B_0\text{-}3C_0$ and $s_{d5} = 3(B_0)\text{-}0(A_1,B_1)\text{-}3(C_0) = 3(B_0)\text{-}3(C_0)$.

iii) The last time looping through Algorithm 3.2, we have $e = C_0$, s_{o5} = empty and s_{d5} = empty.

iv) Algorithm 3.2 returns true.

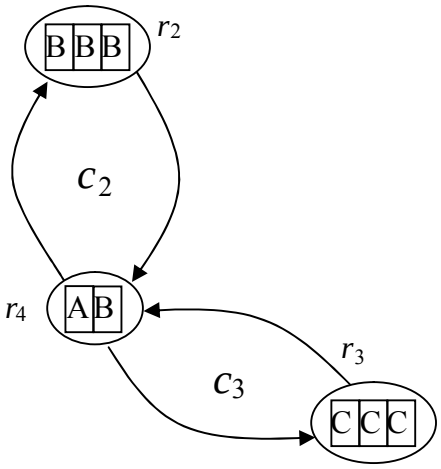

Fig. 4. Deadlock on circuit $c_2 \cup c_3$

With Algorithm 3.1 and 3.2, now we are ready to present our deadlock avoidance supervisor for flexible manufacturing systems.

Deadlock Avoidance Supervisor: Apply algorithm 3.1 to find deadlock string for every circuit (offline). Upon an event e firing request, for each circuit c_i with operating string s_{oi} and deadlock string s_{di}, apply projection $s_o = P(s_{oi}, e)$ and then check if the event is an entering event of the circuit, that is, $e \in E_{en}(c_i)$ and apply Algorithm 3.2 with s_o and s_{di}, if Algorithm 3.2 returns true, reject the event firing request and keep s_{oi} unchanged. If every circuit is checked and no one is rejected, then the event firing request is allowed and s_{oi} is updated to s_o.

The system under the control of the supervisor is called the closed-loop system of the original system. The general closed-loop system architecture is shown in Fig. 5. Notice that the supervisor runs in an incremental mode, that is, the system starts from the empty state and if the current state is not in deadlock, then upon an event firing request, the supervisor determines whether the event firing generates a deadlock and reject or grant the request respectively. For implementation efficiency, the supervisor needs to perform this check only if an event e is an entering event of a circuit c, that is, $e \in E_{en}(c)$, since only an entering event of a circuit can deadlock it. Let $E_{en} = \cup_i E_{en}(c_i)$, then only if $e \in E_{en}$, it is necessary for the supervisor to perform this check. Also, if the check is necessary, then circuits should be checked in the order from small to large.

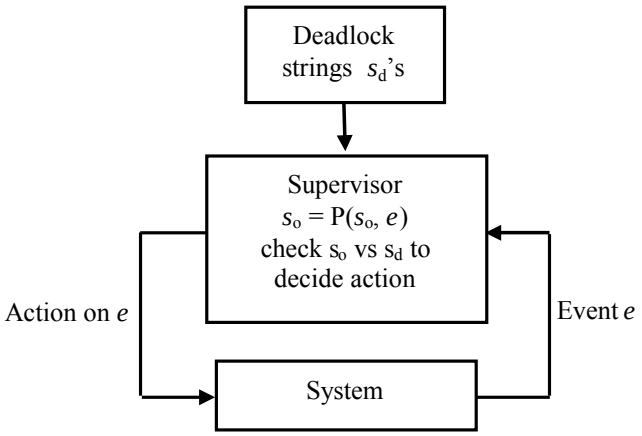

Fig. 5. Closed-loop system diagram

Theorem 3.1: The operating string of a circuit c matching the deadlock string is a necessary and sufficient condition for a system deadlock.

Proof: According to the definition of a deadlock string, once it is reached, none of the underlying jobs on the circuit can exit it and thus form a circular wait situation in which there exists a group of jobs and every job requests a resource that is held by another job in the group. On the other hand, if the operating string does not match any deadlock string, then either an empty resource exists on the circuit or there are events in $E_{ex}(c)$ that means jobs can exit the circuit. In the former case, jobs can move into the empty resource in turn without creating any deadlock since regular systems do not exhibit second level deadlock, until they reach at an exiting point to exit the circuit. Therefore, there will be no deadlock. ■

Theorem 3.2: The closed-loop system under the deadlock avoidance supervisor is live.

Proof: Because the supervisor is applicable to the class of regular systems that do not have second level deadlocks or any higher level deadlocks. The deadlocks corresponding to all deadlock strings are the only deadlocks possible in the system. Hence, when the supervisor exhaustively checks deadlock strings of all circuits, it avoids all deadlocks and allows all live states. ■

Theorem 3.3: The deadlock avoidance supervisor is maximally permissive.

Proof: The proof can be similarly arranged as that of Theorem 3.2. ■

Another good property of the supervisor is the computational efficiency once all circuits are given or calculated offline.

Theorem 3.4: The deadlock avoidance supervisor runs in polynomial time with respect to the number of circuits and the system capacity.

Proof: Let $|s_d|$ be the length of deadlock string s_d that is limited by the system capacity C_R. Let N_C be the number of all circuits. It is in the order of $|s_d|^2$ to check if an operating string matches a deadlock string and it needs to check for N_C times. The projection operation on the operating string is trivial and can be omitted compared to the above checking. Therefore, the supervisor runs in polynomial time in the order of $N_C[\max(|s_d|)]^2 < N_C C_R^2$. ■

Note that N_C should not be too large, since for a normal flexible manufacturing system, the system graph is usually sparsely connected. The efficiency of the supervisor comes from the compound event representation of deadlock strings. Especially for those large circuits, the number of deadlocks on a circuit grows very fast depending on the capacity of the resources and the number of job types sharing the resources of the circuit. Normal event representation would require checking operating strings and every deadlock string combination that could exponentially increase. This state space explosion problem is smartly avoided partially by the proposed deadlock avoidance supervisor. The following example demonstrates the above idea.

4. EXAMPLE AND DISCUSSIONS

In order to show the effectiveness of the proposed deadlock avoidance algorithm, simulation has been run to calculate the state space allowed by the deadlock avoidance method on several examples. Simulation results show that the deadlock avoidance method is indeed correct.

Example 4.1: Consider a simplified track system as shown in Fig. 6. There are six resources r_1~r_6 with various capacities, $C_{r1} = 2$, $C_{r2} = 4$, $C_{r3} = 2$, $C_{r4} = 3$, $C_{r5} = 2$, and $C_{r6} = 4$, resulting in $C_R = 17$. Four types of jobs are defined as A, B, C and D.

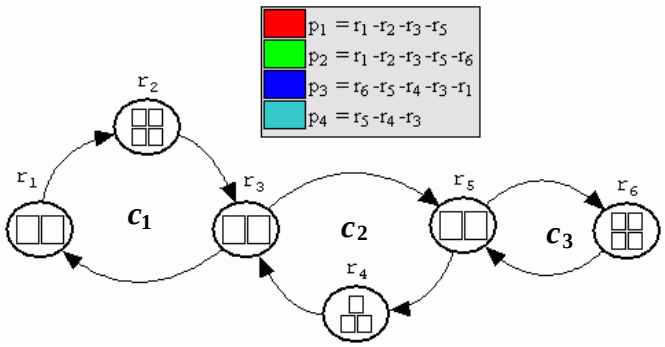

Fig. 6. System graph for example 4.1

The respective event sequences are: $A_0A_1A_2A_3A_4$, $B_0B_1B_2B_3B_4B_5$, $C_0C_1C_2C_3C_4C_5$, and $D_0D_1D_2D_3$. Constructed from all four job types, the system graph consists of total 6 circuits, $c_1 = r_1$-r_2-r_3-r_1, $c_2 = r_3$-r_5-r_4-r_3, $c_3 = r_5$-r_6-r_5, $c_4 = c_1 \cup c_2$, $c_5 = c_2 \cup c_3$, $c_6 = c_1 \cup c_2 \cup c_3$.

It is straight forward to find $E_{en}(c)$ and $E_{ex}(c)$ for all circuits:

i) $E_{en}(c_1) = \{A_0, B_0, C_3\}$, $E_{ex}(c_1) = \{A_3, B_3, C_5\}$.

ii) $E_{en}(c_2) = \{A_2, B_2, C_1, D_0\}$, $E_{ex}(c_2) = \{A_4, B_4, C_4, D_3\}$.

iii) $E_{en}(c_3) = \{B_3, C_0\}$, $E_{ex}(c_3) = \{B_5, C_2\}$.

iv) $E_{en}(c_4) = \{A_0, B_0, C_1, D_0\}$, $E_{ex}(c_4) = \{A_4, B_4, C_5, D_3\}$.

v) $E_{en}(c_5) = \{A_2, B_2, C_0, D_0\}$, $E_{ex}(c_5) = \{A_4, B_5, C_4, D_3\}$.

vi) $E_{en}(c_6) = \{A_0, B_0, C_0, D_0\}$, $E_{ex}(c_6) = \{A_4, B_5, C_5, D_3\}$.

So,

$$E_{en} = \cup_i E_{en}(c_i)$$
$$= E_{en}(c_1) \cup E_{en}(c_2) \cup E_{en}(c_3) \cup E_{en}(c_4) \cup E_{en}(c_5) \cup E_{en}(c_6)$$
$$= \{A_0, A_2, B_0, B_2, B_3, C_0, C_1, C_3, D_0\}$$

Then only if the requested event is in E_{en}, the supervisor needs to check every circuit's operating string against deadlock string for deadlock.

To find the deadlock string for circuit c_1, it is needed to fill up c_1 with all possible jobs and they are, two jobs of type A or B in r_1 correspond to active event $2(A_1,B_1)$, four jobs of type A or B in r_2 – active event $4(A_2, B_2)$, and two jobs of type C in r_3 – active event $2(C_4)$. Then the deadlock string s_{d1} can be found as the causing events of these active events, that is, s_{d1} =

$2(A_0,B_0)$-$4(A_1,B_1)$-$2(C_3)$. Applying Algorithm 3.1 to all 6 circuits will find 6 deadlock strings:

$s_{d1} = 2(A_0,B_0)$-$4(A_1,B_1)$-$2(C_3)$,

$s_{d2} = 2(A_2,B_2)$-$3(C_2,D_1)$-$2(C_1,D_0)$,

$s_{d3} = 2(B_3)$-$4(C_0)$,

$s_{d4} = 2(A_0,B_0)$-$4(A_1,B_1)$-$2(A_2,B_2,C_3)$-$3(C_2,D_1)$-$2(C_1,D_0)$

$s_{d5} = 2(A_2,B_2)$- $2(C_2,D_1)$-$2(B_3,C_1,D_0)$-$4(C_0)$

$s_{d6} =$ $2(A_0,B_0)$-$4(A_1,B_1)$-$2(A_2,B_2,C_3)$-$3(C_2,D_1)$-$2(B_3,C_1,D_0)$ -$4(C_0)$

Deadlock string s_{d1} represents 15 actual deadlocks as given below,

$$\begin{pmatrix} 2A_0 \\ A_0\text{-}B_0 \\ 2B_0 \end{pmatrix} - \begin{pmatrix} 4A_1 \\ 3A_1\text{-}B_1 \\ 2A_1\text{-}2B_1 \\ A_1\text{-}3B_1 \\ 4B_1 \end{pmatrix} - \begin{pmatrix} 2C_3 \end{pmatrix}$$

$$3 \quad \times \quad 5 \quad \times \quad 1 \quad = \quad 15$$

To avoid these 15 deadlocks, whenever there is an event firing request $e \in E_{en}$, the supervisor checks the operating string of circuit c_1 to see if it matches any 2 events of (A_0, B_0) and any 4 events of (A_1, B_1) and 2 events of (C_3) to declare a deadlock. For example, in Fig. 7(a), the operating string of circuit c_1 is $s_{o1} = 1A_0$-$(3A_1,1B_1)$-$2C_3$, loading another job of either type A or B into r_1 of c_1, that is, request to fire either event A_0 or B_0 will be rejected by the supervisor, since $s_o = P(s_{o1}, A_0) = 2A_0$-$(3A_1,1B_1)$-$2C_3$ would match s_{d1}, so would $s_o = P(s_{o1}, B_0) = (A_0,B_0)$-$(3A_1,1B_1)$-$2C_3$.

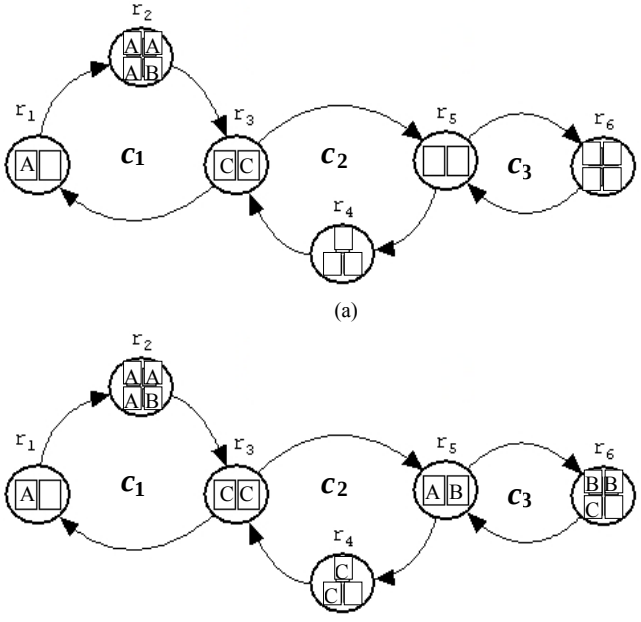

Fig. 7. Deadlock state illustration

In addition, a huge number of deadlocks corresponding to operating strings containing any 2 of (A_0, B_0) and any 4 of (A_1, B_1) and 2 of (C_3) are also avoided by the same check. These deadlocks correspond to all the reachable states that have possible combinations of jobs across r_4, r_5 and r_6, in addition to the parts in r_1, r_2 and r_3, which actually form the deadlock string s_{d1}. For example, in Fig. 7(b), loading another job, either A or B, into r_1 of c_1, that is, a request to fire either event A_0 or B_0 will be rejected by the supervisor.

Hence, by checking the operating string s_{o1} against the deadlock string s_{d1} of circuit c_1, the number of deadlocks avoided by the supervisor is 15 times the total number of possible combinations of jobs across r_4, r_5 and r_6.

For the largest circuit c_6, the deadlock string s_{d6} represents a much bigger number of deadlocks,

$$\begin{pmatrix} 2A_0 \\ A_0\text{-}B_0 \\ 2B_0 \end{pmatrix} - \begin{pmatrix} 4A_1 \\ 3A_1\text{-}B_1 \\ 2A_1\text{-}2B_1 \\ A_1\text{-}3B_1 \\ 4B_1 \end{pmatrix} - \begin{pmatrix} 2A_2 \\ 2B_2 \\ 2C_3 \\ A_2\text{-}B_2 \\ A_2\text{-}C_3 \\ B_2\text{-}C_3 \end{pmatrix} - \begin{pmatrix} 3C_2 \\ 3D_1 \\ 2C_2\text{-}D_1 \\ C_2\text{-}2D_1 \end{pmatrix} - \begin{pmatrix} 2B_2 \\ 2C_1 \\ 2D_0 \\ B_3\text{-}C_1 \\ B_3\text{-}D_0 \\ C_1\text{-}D_0 \end{pmatrix} - \begin{pmatrix} 4C_0 \end{pmatrix}$$

$$3 \quad \times 5 \quad \times 6 \quad \times 4 \quad \times 6 \quad \times 1 = 2160$$

To avoid these 2160 deadlocks, the supervisor checks the operating string of the circuit to see if it matches any 2 events of (A_0, B_0), any 4 events of (A_1, B_1), any 2 events of (A_2, B_2, C_3), any 3 events of (C_2, D_1), any 2 events of (B_3, C_1, D_0) and 4 C_0's to declare a deadlock. This really shows that the supervisor avoids thousands of deadlocks by checking a single deadlock string.

In this example, the system's state space has a total of 4,918,040 states, among which there are 136,720 deadlock states. Simulation results show that indeed the supervisor avoids all 136,720 deadlocks by checking the only 6 deadlock strings and allows all 4,781,320 live states. We also conducted more simulations. Simulation of the illustrative example in Fig. 2 shows that the 300 deadlock states out of the total 9670 states are all avoided by checking 7 deadlock strings. Simulations were also run on other randomly selected regular systems with satisfactory results.

5. CONCLUSIONS

In order to reduce the throughput time and increase the equipment utilization, modern semiconductor manufacturing systems are becoming highly complex with massive automated and flexible equipment. The deadlock issue emerges as a serious obstacle to reliability and productivity. In this paper, a highly efficient and event-based deadlock avoidance supervisor is presented. Concepts such as compound events, string/generalized strings, operating strings and deadlock strings are introduced. Algorithms for finding the deadlock string for a circuit and for matching two generalized strings are provided. The supervisor smartly avoids the deadlock state space explosion problem based on the compound event concept. The method is built upon a

directed graph model of process flows. The supervisor provides the optimal deadlock free operation for a large class of systems. The online computation of the supervisor can be done in polynomial time once the set of deadlock strings are computed offline. Efficiency of the method is demonstrated by simulation results. In the future research, we will extend our results to general systems, where second level deadlocks [5] or more general impending deadlocks [14, 15] exist. And we will extend the result to systems allowing choices in process flows which are not uncommon in semiconductor fabrication.

REFERENCES

[1] Banaszak, Z. and Krogh, B. 1990, "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows," IEEE Trans. on Robotics and Auto., vol. 6, no. 6, pp. 724-733.

[2] Barkaoui, K., and Abdallah, I. B. 1995, "Deadlock Avoidance in FMS Based on Structural Theory of Petri Nets," IEEE Symposium On Emerging Technologies and Factory Automation, V. 2, pp. 499-510.

[3] Cho, H., Kumaran, T.K. and Wysk, R. 1995, "Graph-Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems," *IEEE Trans. on Robotics and Auto*., vol. 11, no. 3, pp. 413-421.

[4] Ezpeleta, J., Colom, J. M. and Martinez, J. 1995, "A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems," *IEEE Transactions on Robotics and Automation*, V. 11, N. 2, pp. 173-184.

[5] Fanti, M.P., Maione, B., Mascolo S., and Turchiano, B. 1997, "Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems", IEEE Trans. on Robotics and Auto., Vol. 13, no. 6, pp. 347-363.

[6] Johnson, D. B. 1975, "Finding All The Elementary Circuits Of a Directed Graph", SIAM J. of Computing, Vol. 4, No. 1, pp. 77-84.

[7] Judd, R. P. and Faiz, T. 1995, "Deadlock Detection and Avoidance for a Class of Manufacturing Systems," *Proceedings of the 1995 American Control Conference*, pp. 3637-3641.

[8] Srinivasan, R. S. 1998, "Modeling and Performance Analysis of Cluster Tools Using Petri Nets", *IEEE Trans. on Semiconductor Manuf.*,, Vol. 11, No. 3, pp. 394-403.

[9] Viswanadham, N., Narahari, Y. and Johnson, T. 1990, "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models," *IEEE Trans. on Robotics and Auto*., vol. 6, no. 6, pp. 713-723.

[10] Wysk, R., Yang, N. and Joshi, S. 1991, "Detection of Deadlocks in Flexible Manufacturing Cells", IEEE Trans. on Robotics and Automation, Vol.7, No.6, pp.853-859.

[11] Xing, K., Hu, B. and Chen, H. 1996, "Deadlock Avoidance Policy For Petri-Net Modeling Of Flexible Manufacturing Systems With Shared Resources," *IEEE Transactions on Automatic Control.*, vol. 41, no. 2, pp. 289-295.

[12] Yalcin, A. and Boucher, T. 2000, "Deadlock Avoidance in Flexible Manufacturing Systems Using Finite Automata", *IEEE Trans. on Robotics and Auto*., vol. 16, no. 4, pp. 424-429.

[13] Yoon, H. J. and Lee, D. Y. 2000, "Deadlock-Free Scheduling Method for Track Systems in Semiconductor Fabrication," Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics, Nashville, Tennessee.

[14] Zhang, W., Judd, R. P. and Deering, P. 2004, "Necessary and Sufficient Conditions For Deadlocks in Flexible Manufacturing Systems Based on a Digraph Model", *Asian Journal of Control*, Vol. 6, No. 2, pp. 217-228.

[15] Zhang, W., Judd, R. P. and Paul, P. 2003, "Evaluating Order of Circuits for Deadlock Avoidance in a Flexible Manufacturing System", *Proceedings of the 2003 American Control Conference*, Denver, pp. 3679-3683.

[16] Zhou, M. and DiCesare, F. 1992, "Parallel and Sequential Mutual Exclusion for Petri Net Modeling of Manufacturing Systems with Shared Resources," *IEEE Trans. on Robotics and Auto*., vol. 7, no. 4, pp. 515-527.

[17] Zhou, M. C. and Jeng, M. 1998, "Modeling, Analysis, Simulation, Scheduling and Control of Semiconductor Manufacturing Systems: A Petri Net Approach", *IEEE Trans. on Semiconductor Manuf.*, Vol. 11, No. 3, pp. 333-357.

Wenle Zhang received his B.S. and M.S. from Shandong Univ. of Sci. and Tech. in 1983 and Shanghai Univ. of Sci. and Tech. in 1986, respectively. Then He worked as an instructor in the dept. of computer engineering at Shanghai Univ. of Sci. and Tech. until 1990. He received his PhD in Electrical Engineering from Ohio University in 2000. After received his PhD, Dr. Zhang worked for Lucent Technologies as a software engineer until March 2001. Prior to pursuing his PhD at Ohio University, he was employed with Rockwell Automation, a famous industrial controls supplier, as a control engineer for more than 5 years. He has experience in PLC application, industrial control networking, and control software development.
Since spring of 2001, Dr. Zhang has been an assistant professor in the School of Electrical Engineering and Computer Science at Ohio University. Dr. Zhang's current research interests include manufacturing system, industrial controls, distributed computing, neural networks and intelligent control.

Ziqiang J. Mao received the B.S. degree in mathematics and the M.S. degree in computer and system sciences from Nankai University, China in 1983 and 1986 respectively. He also received the Ph.D. degree from the University of California, Davis, in 1995. Since 1994, he has been with Intel Corporation. In Intel, he is a technologist and research scientist and works on managing the process control program, developing process control systems and providing strategic directions, teaching engineers and conducting researches in process control, prediction and monitoring. He has published 36 papers in journals and conferences in the areas of VLSI design, computer graphics, robotics, manufacturing scheduling, statistical process control, multivariate fault detection and classification, process run-to-run control, and multi-step cooperative process control. He is an IEEE senior member.