Synthesis of An Event Based Supervisor For Deadlock Avoidance In Semiconductor Manufacturing Systems

† Wenle Zhang
School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio 45701

Ziqiang John Mao
Intel Corporation
California Technology Manufacturing
2200 Mission College Blvd.
Santa Clara, CA 95052

Abstract With the emerging of highly automated and flexible manufacturing systems in semiconductor fabrication, reliability and optimal productivity of such systems require very intelligent and complex control systems. Deadlock issue arises easily in these systems due to shared equipment usage and high production flexibility. This paper presents a new event-based deadlock avoidance supervisor. The supervisor 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, operation strings and deadlock strings are introduced. Major features in the proposed method includes: i) it enables the 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.

Keywords: semiconductor manufacturing system, discrete event, digraph, circuit, deadlock avoidance.

1. Introduction

Currently productivity and equipment utilization is critical in semiconductor industry to reduce the cost of product and support decreasing trend of average sale price (ASP). The goal of manufacturing scheduling and deadlock avoidance is to ensure an optimal solution for the manufacturing operations that maximizes product output, equipment utilization and availability; and minimizes the wafer process throughput-time and cost. Optimal scheduling of these working entities can have as a big impact at millions of dollars per year.

The manufacturing operation is a process with equipment, wafers, and technicians. At each step, if any one of them is not available, a wafer processing step cannot continue. Many problems exist in manufacturing, such as, i) equipment waits for wafers; ii) wafers wait for equipment; iii) tool waits for a technician's response (setup, un-loading, assist, etc.); iv) wafers and technicians are available, but tool is idle and waits for loading. For example, the multi-million dollar lithographic equipment is the key bottleneck components in a fab. The wafers to be processed in photolithography step come from different process functional areas such as diffusion, thin film, polish etc. The optimized scheduling of wafer movement and litho equipment utilization are critical to the total throughput. Also, the wafers from different areas into one area can encounter traffic jam and slow down the process flow.

One way to minimize the throughput time is to increase the quantity of capital. After increasing the number of very expensive equipments, more equipment idle time is observed. A fab (fabrication site) is always equipped with certain number of similar equipment. For example, a Fab may have 10-20 lithography steppers/scanners. So the minimization of throughput time and minimization of equipment idle time and cost requires a balanced optimization on both. Also, with 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 [16] 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 [9]. 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 work have adopted Petri net (PN) models [1,2,4,10,12,15] as a formalism to describe the manufacturing system. Banaszak[1] proposed a *deadlock avoidance algorithm* (DAA) that developed a restriction policy to guarantee that no circular wait situations will occur. Viswanadham [10] developed a deadlock avoidance algorithm which suggested using a recovery mechanism in case of system deadlock. Zhou [15] 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 in [2,4] to determine potential deadlock situations.

Another formalism is to describe the manufacturing system using graphs [3,5,7-8,11,14]. In this approach the vertices represent resources and the arcs (edges) represent product part flows between resources. Cho [3] developed the concept of bounded circuits with empty and non-empty shared resources to detect deadlock. Judd [7] derived a set of static linear inequalities that when they are satisfied deadlock is avoided. Lipset [8] expanded upon [7] and quantified both necessary and sufficient conditions for deadlock to occur in a manufacturing system.

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 new high 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. Concepts such as compound events, operation strings and deadlock strings are introduced. There are two major features in the proposed method. Firstly, 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 Xing [12].

† Corresponding author. Phone: 740-597-1481, Fax: 740-593-0007, Email: zhangw@bobcat.ent.ohiou.edu

Secondly, it runs in polynomial time, which means faster online computation, once the set of deadlock strings are computed offline. Examples 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 is 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 be performed on exactly one resource. Thus a process p can be represented as a sequence of resources $p=r_1r_2\ldots r_m$. Each resource $r \in R$ has a *capacity*, denoted as C_r (>1 for regular systems), which can be considered as buffers at the resource or a multiple of 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 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, $G' = \text{trim}(G)$ consists of all isolated SCCs. In the following, we will work on G'.

2.1. Conceptual System Deadlock

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

Type A wafer: Tool1 – Robot – Tool2
Type B wafer: Tool2 – Robot – Tool3
Type C wafer: Tool3 – Robot – Tool1

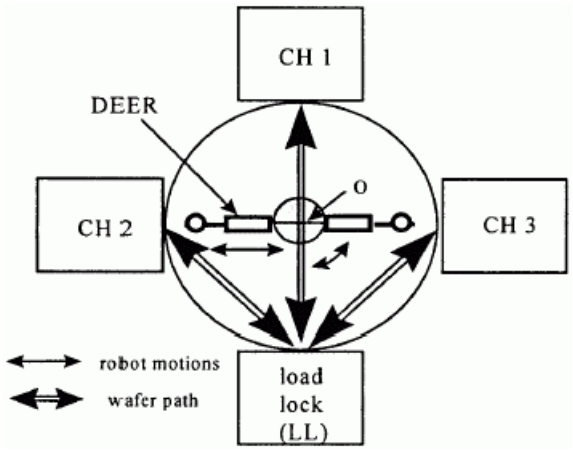

Figure 1. Cluster tool configuration

Assume the system is fully automated and Tool1, Tool2 and Tool3 have capacity 3, Robot has capacity 2. And a finished product leaves the system automatically. The system graph is constructed in figure 2. The graph itself is a SCC.

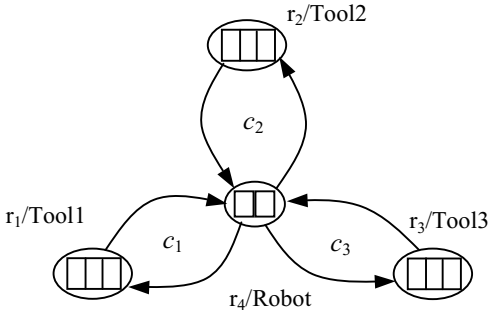

Figure 2. Simplified directed graph

Then, we can observe 6 deadlocks:

i) when r_1 has 3 type A wafers and r_4 has 2 type C
ii) when r_2 has 3 type B wafers and r_4 has 2 type A
iii) when r_3 has 3 type C wafers and r_4 has 2 type B
iv) when r_1 has 3 type A wafers, r_2 has 3 type B wafers and r_4 has 1 type A and 1 type C
v) when r_2 has 3 type B wafers, r_3 has 3 type C wafers and r_4 has 1 type B and 1 type A
vi) when r_3 has 3 type C wafers, r_1 has 3 type A wafers and r_4 has 1 type C and 1 type B

Also, if the Robot has capacity 3, then an extra deadlock is as r_1 with 3 type A wafers, r_2 with 3 type B wafers, r_3 with 3 type C wafers, all at their first process step and Robot has 1 type A, 1 type B and 1 type C. Observe that each deadlock is related to a circuit that is the root cause of a circular wait situation. E.g., the first deadlock corresponds to $c_1 = r_1\text{-}r_4\text{-}r_1$, the forth deadlock relates to $c_1 \cup c_2 = r_1\text{-}r_4\text{-}r_2\text{-}r_4\text{-}r_1$.

Formally, deadlock is defined as a circular wait situation [1] in which there exists a group of jobs and every job requests a resource that is held by another job in the group.

2.2. System Events

Once the system is in operation, the system will evolve with the occurrences of system events, such as loading a job into the system, transporting a job from one step to next step, or unloading a finished job from the system. In this paper, we consider only these high level events since they are most related to deadlock occurrences. So, each process type 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 type.

Each event e has a *source-step* and a *destination-step* , denoted as *e.ss* and *e.ds*, respectively, 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 figure 2, if we name events corresponding to the three types of jobs with A, B and C, then type 1 job has event sequence $A_0A_1A_2A_3$, $A_0.ss = 0$, $A_0.ds=1$, $A_2.ss = 2$, 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 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.

Obviously, events with the same type have to occur in the order described by the process type. For example, A_0 has to occur first in order to have an active A_1 event. We say that 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 observed 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 [11][6]. In the following, we assume all circuits are given.

A sequence of events is called a *string*, such as string $s = A_0$-B_1-B_2-A_1. 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 whose destination-step matches the resource or the source-step matches the resource. E.g., r_1's shared events set: $\{A_0\ C_2\}$ and $\{A_1\ C_3\}$, called *enter event set* , denoted as $E_{en}(r_1)$ and *exit event set*, denoted as $E_{ex}(r_1)$ of the resource, respectively. All such sets can be obtained by analyzing process types. After event A_0 of $E_{en}(r_1)$ occurs, the underlying job will be in the resource r_1 and after event A_1 of $E_{ex}(r_1)$ occurs, the same job is out of r_1.

A *compound event* is defined on a resource as a number (less than the capacity of the resource) followed by a subset of a shared event set of the resource. E.g., for r_1, we have $2(A_0)$ or $2(A_1,C_3)$, where an active compound event $2(A_1,C_3)$ means 2 active A_1 events, 1 active A_1 event and 1 active C_3 event, or 2 active C_3 events. A normal event e can then be considered as a compound event in the form of $1(e)$. From now on, both normal events and compound events are simply referred to as events.

A *generalized string* is defined to be a sequence of compound events.

As mentioned earlier, deadlock is related to circuits. Enter/exit event sets can be extended to a circuit, e.g., 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 1: An *operating string* of a circuit is the sequence of events that have occurred so far on the circuit. E.g., the operating string for c_1 after some time is $s_o = 2(A_0)$-A_1-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 type A job at r_4 and one type A job at r_3.

Definition 2: 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.

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. For example in figure 2, when circuit c_1 is in deadlock (the first deadlock), then 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, for the example, the deadlock string in this case is $s_d = 3(A_0)$-$2(C_1)$.

Algorithm 1**:** DS **----** Finding the deadlock string for a circuit $c = (R_1, A_1)$

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

Output: s_d – deadlock string of c

```
First find Eex(c)
For each r in R1
  Er = {} - set of event sharing resource r
  For each process type p in P
    For each event e in EP
      Let re = p(e.ss) be the resource at
        step e.ss
      If r == re and e not in Eex(c)
        Add e to Er
    End for
  End for
  Append compound event Cr(Er) to sd
End for
Sd = translate each event of sd 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.

Define an *ss* (*ds*) operation on a string s as a mapping: $ss(s) \rightarrow N^{|s|}$, that is, to give the set of source (destination)-step of all events in the string. E.g., if $s_o = 2(A_0)$-A_1-A_2, then $ss(s_o) = \{0, 0, 1, 2\}$.

Projection Operation: Given a circuit c, the deadlock string s_d of c and an event e, define a projection operation on the operating string s_o of c such that,

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

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, property i) means to remove the first causing event of e from s_o and property ii) means if event e exits the circuit, then it should not stay in the operating string.

Deadlock Avoidance Supervisor: Apply algorithm 1 to find deadlock string for every circuit (offline). Upon an event e firing request, for each circuit in the order from small to large circuits, apply projection $s_{o1} = P(s_o, e)$ and then check if the event is an entering circuit event and $|s_{o1}| < |s_d|$ or s_{o1} does not match the deadlock string s_d of the circuit, then accept the event firing request and set s_o to s_{o1}, otherwise reject the event firing request and keep 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 figure 3.

Theorem 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 deadlock string, once it is reached, none of the underlying jobs on the circuit can exit the circuit and 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 the deadlock string, then either there exists empty resource on the circuit or there exists events in $E_{ex}(c)$ that means jobs can exit the circuit. In the former case, jobs can move into empty resource in turn until they reach at an exiting point to exit the circuit. Therefore, there will be no deadlock. ■

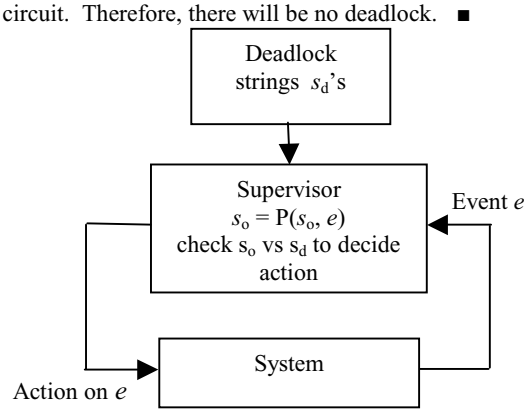

Figure 3. Closed-loop system diagram

Theorem 2: The closed-loop system under the deadlock avoidance supervisor is live.

Proof: The essential reason is the class of regular systems that the supervisor applies does not have second level deadlock or any other impending deadlock. The deadlocks corresponding to all deadlock strings are the only deadlocks possible in the system. Then if the supervisor exhaustively checks deadlock string of each circuit, it will avoid all deadlocks and allow all live states. ■

Theorem 3: The deadlock avoidance supervisor is maximally permissive.

Proof: The proof can be similarly drafted as that of theorem 2.

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

Theorem 4. The deadlock avoidance supervisor runs in polynomial time.

Proof: Checking for a match of an operating string and a deadlock string is in the order of $|s_d|^2$, where $|s_d|$ is the length of the deadlock string that is limited by the system size (the total resource capacity). And such checking needs to be done for N_C, number of all circuits, times. Also compared to the above checking, the trivial efforts of the projection operation on the operating string can be omitted. Therefore, the supervisor runs in polynomial time in the order of $N_C[\max(|s_d|)]^2$. ■

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, number of total 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 string and every deadlock string combination that could exponentially increase. This state space explosion problem is smartly avoided by our deadlock avoidance supervisor. The following example demonstrates the above idea.

Example 1. A simplified track system is shown in figure 4. Four types of jobs are defined as p1-p4. 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 $D_0D_1D_2D_3$. The system graph consists of total of 6 circuits, $c_1 = r_1\text{-}r_2\text{-}r_3\text{-}r_1$, $c_2 = r_3\text{-}r_5\text{-}r_4\text{-}r_3$, $c_3 = r_5\text{-}r_6\text{-}r_5$, $c_4 = c_1 \cup c_2$, $c_5 = c_2 \cup c_3$, $c_6 = c_1 \cup c_2 \cup c_3$. It is straight forward to find $E_{en}(c)$ and $E_{ex}(c)$ for all circuits c. Such as, $E_{en}(c_1) = \{A_0, B_0, C_3\}$, $E_{ex}(c_1) = \{A_3, B_3, C_5\}$.

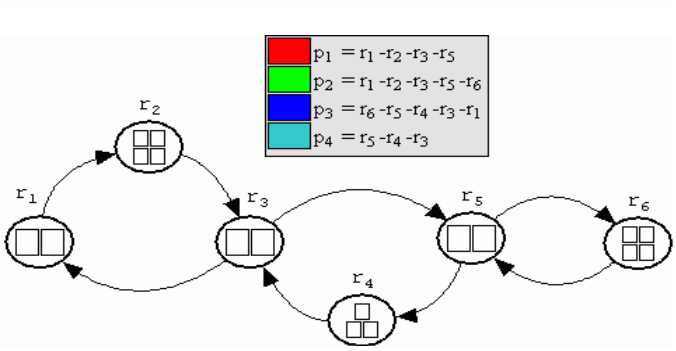

Figure 4. System graph for example 1

Applying algorithm 1 will find 6 deadlock strings:

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

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

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

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

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

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

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

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

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

To avoid these 15 deadlocks, the supervisor checks the operating string of the circuit to see if it matches any 2 of (A_0, B_0) and any 4 of (A_1, B_1) and 2 of (C_3) to declare a deadlock. 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 parts across r_4, r_5 and r_6, in addition to the parts in r_1, r_2 and r_3, which actually formed the deadlock string s_{d1}.

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

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

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

To avoid these 2160 deadlocks, the supervisor checks the operating string of the circuit to see if it matches any 2 of (A_0, B_0), any 4 of (A_1, B_1), any 2 of (A_2, B_2, C_3), any 3 of (C_2, D_1), any 2 of

(B_3, C_1, D_0) and 4 C_0's to declare a deadlock. This best 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. Simulation on the conceptual example in figure 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 random selected regular systems and the results are satisfactory.

4. Conclusions

In order to reduce the throughput time and increase the equipment utilization, modern semiconductor manufacturing system is 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. The supervisor smartly avoids the deadlock state space explosion problem based on the compound events 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. In the future research, we will extend our results to irregular systems, such as the cluster tool where the center robot has a single blade (SEER) as opposed to figure 1 double blades (DEER), where second level deadlock [5] or more general impending deadlock [8,14] exists. And we will extend the result to systems allowing choices in process flows which are not uncommon in semiconductor fabrication.

5. References

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

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

[3] Cho, H., T.K. Kumaran, and R. Wysk, "Graph-Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems," *IEEE Trans. on Robotics and Auto*., vol. 11, no. 3, pp. 550-527.

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

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

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

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

[8] Lipset, R., P. Deering, and R. P. Judd, "Necessary and Sufficient Conditions for Deadlock in Manufacturing Systems," *Proceedings of the 1997 American Control Conference*, vol. 2, pp. 1022-1026, June 1997, Albuquerque.

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

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

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

[12] Xing, K., B. Hu and H. Chen, "Deadlock avoidance policy for Petri-net modeling of flexible manufacturing systems with shared resources," *IEEE Transactions on Automatic Control.*, vol. 41, no. 2, 1996, pp. 289-295.

[13] Yoon, H. J. and D. Y. Lee, "Deadlock-Free Scheduling Method for Track Systems in Semiconductor Fabrication," Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics, Nashville, Tennessee, USA, October 8-11, 2000.

[14] Zhang, W., R. P. Judd and P. Paul, "Evaluating Order Of Circuits For Deadlock Avoidance In A Flexible Manufacturing System", *Proceedings of the 2003 American Control Conference*, pp. 3679-3683, June 2003, Denver.

[15] Zhou, M. and F. DiCesare, "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, 1992, pp. 550-527.

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