RECIPE CASCADING IN A WAFER PROCESSING SYSTEM

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the field of wafer processing. In particular, the invention relates to scheduling techniques for wafer cluster tools.

Description of the Related Art

In the process of manufacturing a semiconductor device such as an integrated circuit, numerous steps of micro-fabrication are performed to form a device. These steps are performed serially on the individual items of manufacture in individual modules; the items of manufacture are transferred between modules by transport mechanisms such as robots. In order to achieve desirable throughput, reliability, and fabrication quality, several conditions must be met:

1) The delivery and removal of the substrate to and from the process modules, as well as the transportation of the wafer between modules, must be accomplished in a timely manner. This timely delivery and removal of substrate is achieved when the flow of substrate is maintained in a periodic and synchronized manner. If periodicity and synchronization are not maintained, the process results will be inconsistent from substrate to substrate, and the expected throughput may be reduced.

2) It is desirable to transport the substrate in similar process flow paths to avoid inconsistency in process results due to variations in the process history of the substrates.

3) It is imperative to ensure that the articles of manufacture do not spend any pre-process or post-process time idling in modules where critical processes are performed. The addition of pre-process or post-process time in these modules degrades not only the throughput but also the process results. For example, in an IC fabrication system, if a substrate is not immediately transferred from the spin

coat module to a bake module to thermally cure a photo-resist film layer, the resulting film thickness will be unpredictable. If it is impossible to totally eliminate pre-process and/or post-process times, they should be rendered as brief as possible, and any variations in these times cannot be allowed.

The inability to meet any or all of the above conditions come from the failure to resolve transport conflicts. Conflicts are situations wherein separate modules demand a robot within a time span insufficient for the robot to service these modules

One conventional solution to the concerns listed above is the addition of extra process modules and transportation resources. However, the size limitations and geometrical constraints of a track system limit the possibility of resolving the above difficulties by adding additional process modules or transportation resources.

The addition of dedicated transfer arms to transfer substrates between adjacent modules (hereinafter called Inter Bay Transfer Arms, or IBTAs) is another method used to improve throughput and eliminate some of the pre-process and/or post-process times. However, the addition of IBTAs also has serious drawbacks. Dedicated transfer arms complicate the tool and increase its cost, constrain the position of the modules, and cannot be used everywhere in the tool. As a result, the tasks of managing the substrate flow in the track system while maintaining both high throughput and quality and resolving all transport conflicts become unmanageable.

Another conventional solution is to assign a set of substrate transport priority rules. Prior to any robot move, the control system, also referred to as the software scheduler, verifies the status of substrates in different modules and makes transfer priority decisions based on these rules. However, to achieve high throughputs, the scheduler may generate undesirable, unpredictable and variable pre-process and post-process times in critical modules, and the substrates may also be forced to follow different flow paths to complete their process cycle.

Heretofore, the requirements of conflict resolution, synchronization, quality, and path consistency referred to above have not been fully met. What is needed is a solution that simultaneously addresses all of these requirements.

SUMMARY OF THE INVENTION

An embodiment of the invention allows a cluster tool to change from a first recipe to a second recipe, while preserving periodicity and ensuring that there are no delays at critical points. This procedure is referred to as recipe cascading. Cascading involves emptying a first lot of wafers off a cluster tool and populating the cluster tool with another lot of wafers, serially and simultaneously. The procedure is performed with no delays incurred at critical process steps; and with no additional robots and process modules other than those called for by the recipe and throughput requirements of the entering and the exiting lots. The entering lot may also have different recipe and throughput requirements from the exiting lot.

An embodiment of the invention includes a method for processing wafers which comprises loading a first plurality of wafers into a wafer cluster tool individually at intervals delimited by a first sending period, wherein the first plurality of wafers are processed according to a first recipe; and loading a second plurality of wafers into the cluster tool at intervals delimited by a second sending period, wherein the second plurality of wafers are processed according to a second recipe. In embodiments of the invention, the cluster tool has a transition period, during which the cluster tool processes one or more wafers from the first plurality of wafers according to the first recipe and one or more wafers from the second plurality of wafers according to the second recipe.

Embodiments of the invention also include a computer program for scheduling the wafer processing system. The computer program includes resources for scheduling the wafer processing system during a first time period, wherein a first plurality of wafers is processed during the first time period according to a first recipe; resources for scheduling the wafer processing system during a second time period, wherein a second plurality of wafers is processed during the second time period according to a second recipe; and resources for

scheduling the wafer processing system during a third time period, wherein a third plurality of wafers is processed during the third time period, such that one or more wafers from the third plurality are processed according to the first recipe, and one or more wafers from the third plurality are processed according to the second recipe. In embodiments of the invention, the computer program resides on a server coupled to the wafer processing system. In embodiments of the invention, the computer program uses a genetic algorithm to schedule the wafer processing system. In some embodiments, the computer program uses another optimization technique to schedule the wafer processing system.

BRIEF DESCRIPTION OF THE FIGURES

Figure 1 is a time line illustrating the points at which wafers are loaded into a cluster tool, wherein the points are separated by intervals of one sending period.

Figure 2 illustrates the relative and absolute pick-up times of three wafers at a process i, as well as the mantissas of the pick-up times, wherein the pick-up times are normalized in terms of the sending period.

Figure 3 illustrates the various modules and transportation modules in the cluster tool.

Figure 4 is a graph and corresponding table illustrating the module paths in the cluster tool.

Figure 5 is a table illustrating the recipe of a cluster tool.

Figure 6 is a graph and corresponding table of module pick-up times which arise in a sending period, wherein the sending period is broken into 6 sub-intervals, as there are six possible robot moves within a single sending period.

Figure 7 is a graph and corresponding table illustrating conflicts that arise between processes for access to robots, wherein conflicts are indicated when two modules assigned to a robot have pick-up demands within a single period.

Figure 8 is a graph and accompanying tables and matrices, which illustrate the insertion of queues to eliminate conflicts between modules for access to robots.

Figure 9 illustrates recipes and recipe times in a cluster tool.

Figure 10 illustrates wafer flow in steady state in a cluster tool.

Figure 11 illustrates a transition period, in which the cluster tool processes an entering lot and an exiting lot.

Figure 12 illustrates the timing of the first wafer of the entering lot in the cluster tool according to an embodiment of the invention.

DETAILED DESCRIPTION

Synchronized, Conflict Resolving Schedulers

An aspect of the present invention comprises a method for maximizing throughput and quality in a manufacturing system by scheduling events in the system in a periodic, predictable fashion which eliminates conflicts for system resources. An example of such a manufacturing system comprises a series of process steps 1, . . ., N, which are performed consecutively on individual units of manufacture. The individual process steps of the system are conducted in "modules", or "process chambers", and the series of steps is listed in a "recipe". The manufacturing system also includes resources for transporting the units of manufacture between modules in the series; these resources may include robots.

Conflicts may result between processes in the system when separate modules demand a robot within a time span which is insufficient for the robot to service these modules. Additionally, it is desirable to schedule the system in a manner which exhibits periodicity, so that events in the system are synchronized to occur at periodic, predictable intervals. An embodiment of the present invention includes a technique of selectively scheduling delays in various steps of the manufacturing process in order to eliminate all such conflicts, as well enforce periodicity, without degrading throughput or quality of the system.

An Example of Conflict-Resolving Synchronization: Wafer Cluster Tools

An example of the type of manufacturing system described above is a wafer cluster tool. In a wafer cluster tool, the modules comprise process chambers, which are organized around a group of wafer transporting resources, or robots, to perform a sequence of process steps on the wafer. A wafer enters and exits the tool through a buffer called a load port. Once a robot retrieves a

wafer from a load port, the wafer is transported sequentially through the series of modules specified in a recipe. The time period defined by a wafer's entrance to a module and the wafer's exit from the module is referred to as a module process time. This process time includes the time actually spent processing the wafer in the module as well as the overhead time required to prepare the wafer for processing and pick up.

(note that though the wafer cluster tool is described above as passing an individual wafer between modules, it will be apparent to one skilled in the art that the present invention is equally applicable to a wafer cluster tool in which a discrete set of wafers is passed between modules.)

In certain modules of the cluster tool, a delay in picking up the processed wafer may adversely affect on-wafer results; such modules are identified as "critical process modules," as they cannot tolerate delays. The module whose process time is longest amongst all modules in the cluster tool is identified as the "gating module"; the process time at this module determines the throughput of the cluster tool. Because the gating module determines the throughput of the cluster tool, it too cannot tolerate delays. The recipe for a wafer cluster tool lists the modules in sequential order, alongside their respective process times. The time required by a robot to transport a wafer between two modules is referred to as its transport time.

Wafer Flow Management in the Cluster Tool

Wafer flow management, i.e., the orchestration of wafer processing and wafer transporting in a cluster tool, determines both the throughput and the on-wafer results delivered by the system. Effective wafer flow management requires the simultaneous satisfaction of the following two conditions: a wafer which was just processed in the sending module and is now ready to move should do so when (1) the receiving module in which the wafer will subsequently be processed is empty; and (2) the robot assigned to transport wafers between those modules is available. In the prior art, condition (1) was satisfied by providing additional redundant modules. Such a solution, however, compromises condition (2) in two ways: (a) it results in an inadequate number

of robots serving too many modules or (b) two or more modules may compete simultaneously for the service of a robot.

When the two conditions listed above are compromised, delays in wafer pick-up result. If such delays occur at critical process modules, they adversely affect on-wafer results. And if such delays occur at the gating module, they slow down throughput. As such, it is imperative that the transport conditions listed above are guaranteed with respect to critical process modules and the gating module. In case (a) wherein more handling is required than the robots could provide, adding more robots can mitigate the situation. However, in case (b), the problem resides in the timing of the robot service request. While adding more robots can also alleviate case (b), this is an inadequate solution.

Since the recipe prescribed for the cluster tool determines the timings of the robot service request, a fundamental solution to resolving the two conditions can arise from altering the wafer recipe to synchronize with wafer transport. A scheduling algorithm described herein performs such synchronization.

This scheduler described herein can be encoded in software executed by a computer, wherein the computer comprises a memory for storing the software, and a CPU for executing the software. In an embodiment of the present invention, the scheduler may be used off-line from the manufacturing system to generate a pre-determined schedule for the system. Alternatively, the computer may be coupled to the manufacturing system so that the scheduler can update the operation of the system in real-time.

Synchronizing Wafer Flow in the Cluster Tool

In an embodiment of the present invention, wafer flow is synchronized by sending individual wafers through the cluster tool at a constant rate. This rate, referred to as the tool's "sending rate", is expressed in number of wafers per hour, and paces the wafer flow with a periodicity equal to (3600/sending rate) seconds. This period, referred to as the sending period of the cluster tool, is the heartbeat of the system. Individual wafer units are introduced to the system at intervals of one sending period. And in order to synchronize the cluster tool, all process and transport times are measured in units of sending period. Furthermore, to ensure that the same tasks can be repeated in succeeding

periods, the robots in the cluster tool are scheduled to accomplish all service requests, hereafter referred to as "tasks", within a single sending period. As such, the synchronization of the cluster tool requires a determination of 1) the total number of tasks that are performed within a sending period and 2) the exact moment within a sending period that these tasks arise. These moments shall hereafter be referred to as the "timings" of the respective tasks.

The concepts of sending periods and synchronization are illustrated in Figure 1. The timeline 100 has an origin 102, which denotes the moment when the first wafer is loaded into the cluster tool. The timeline 100 is demarcated in units of one sending period 110. Each demarcation 104 106 108 indicates, respectively, the times at which the second, third, and fourth wafers are loaded into the cluster tool.

A principal characteristic of synchronization is periodicity: the present invention ensures that for each task i, i=1, . . ., n, the pick-up times for any wafer undergoing that task are identical. Thus each task i in the cluster tool can be associated with a relative pick-up time denoted T_i, where T_i is normalized in units of the sending period. Figure 2 depicts this feature of periodicity. Three wafers, wafer 1 208, wafer 2 210, and wafer 3 212 are depicted on the vertical axis 202. The horizontal line depicts the TIME axis 200. The origin of this axis 201 indicates the time at which wafer 1 is loaded into the cluster tool. The relative pick-up times T_i at task i 200 are identical for each wafer. Because the wafers themselves are introduced at intervals of one sending period, the actual pick-up times are separated by units of one sending period.

Figure 2 also illustrates a distinction between relative and "actual" or "absolute" pick-up times. The relative pick-up time of a process i is denoted by T_i 204. Since the relative pick-up time is measured from the time a wafer is introduced into the wafer cluster tool, the relative pick-up time is identical for each wafer, wafer 1 208, wafer 2 210, and wafer 3 212. The absolute pick-up time 214 is measured from the moment the first wafer was loaded into the cluster tool 201. Since the wafers are introduced at intervals of one sending period, it follows that for any wafer no. w, the absolute pick-up time of wafer w at module i is

$(w-1) + T_i$

This period (w-1) is illustrated in the figure 216.

Another parameter which is critical in synchronization is designated by the symbol τ_i. The fraction $\tau_i = T_i - INT(T_i)$, where $INT(T_i)$ is a function that rounds T_i down to the nearest integer, is the fraction of T_i that has elapsed since the beginning of the current sending period. These parameters 206 are also illustrated in Figure 2. Since the T_i values are identical for each wafer, and since the wafers are inserted at intervals of one sending period, the values of τ_i 206 are identical for each wafer. These fractions, τ_i, i = 1, 2, 3 . . . N comprise the timings of the tasks the robots must accomplish within a sending period.

The number of tasks N and the timings of these tasks constitute the load of the transport. Since T_i is the accumulation up to the *i*th module of the process times $p_j, j = 1,2,3...i$; and the robot transport times $t_j, j = 1,2,3...i-1$, it follows that for any wafer, the relative pick-up time at module i is:

$$T_i = \sum_{j=1}^{i} p_j + \sum_{j=1}^{i-1} t_j; \qquad i = 1,2,3...N$$

It also follows that the timing of the tasks, $\tau_i = 1,2,3...N$; is

$$\tau_i = T_i - INT(T_i)$$

$$= \sum_{j=1}^{i} p_j + \sum_{j=1}^{i-1} t_j - INT\left(\sum_{j=1}^{i} p_j + \sum_{j=1}^{i-1} t_j\right); \qquad i = 1,2,3...N \qquad (1)$$

Since transport times t_j are fixed for a given cluster tool, it is apparent from Equation (1) that the timing of robot task τ_i is dependent solely on the process times p_i as prescribed by the recipe.

Periodicity and Wafer Identification

The property of periodicity also enables the identification of wafers in the cluster tool. As elaborated infra, the synchronized scheduler ensures that 1) the wafers are loaded into the cluster tool in sequential order at intervals of one sending period, and 2) each wafer loaded into the cluster tool undergoes identical events at the same times, as measured relative to the moment they are loaded. A consequence of these two conditions is that wafers enter and depart

each module in the cluster tool in the order they were originally loaded, at intervals of one sending period. As such, each wafer in a module can be identified simply by tracking the order in which they entered or exited that module. This feature of the synchronized scheduler is referred to as wafer identification, or wafer "tagging".

Tagging and Module Paths

In an embodiment of the present invention, each wafer loaded into the cluster tool follows a particular "module path", i.e., a particular set of modules which correspond to the processes in the cluster tool. This feature of the present invention is illustrated in Figure 4. In this embodiment, each process in the cluster tool has one or more modules associated with it, wherein the wafers are processed. The modules for each process are ordered in a sequence such that when wafers arrive at that process, they are placed in the corresponding modules in the sequential order (for e.g., if a process has two corresponding modules, the first wafer in the system goes to the first module, the second wafer goes to the second module, the third wafer enters the first module, the fourth wafer enters the second module, etc.) As a consequence, the total number of module paths that a wafer may follow is constrained to the least common multiple of the number of modules corresponding to each process.

The embodiment described above is illustrated by example in Figure 4. Figure 4 shows a sequence of process steps, VP 400, VPC 402, CT 404, SB 406, SBC 408, PEB 410, PEBC 412, DEV 414, HB 416, HBC 418. A symbol of a process step appears for each module corresponding to the process step. For instance, the process CT 404 has three modules, and corresponding, the symbol of CT appears three times 404. Above each process step is the number of modules for that process step 420.

In this example, the least common multiple of the number of modules is:
LCM(2,2,3,3,3,3,3,4,2,2) = 12

Hence, the recipe for the cluster tool prescribes twelve module paths, which are listed 422. Each column in the table 422 lists the module number for that process step in the respective module path. As there are twelve possible paths, every twelfth wafer follows the same module path. As such, by

identifying a wafer and the order in which it was loaded into the tool, the present invention enables the determination of the module path followed by the wafer.

Adding Queues to Eliminate Conflicts for Transportation Resources

If a recipe gives rise to simultaneous, competing service requests for particular robots, it would be desirable to resolve the conflicts not by adding more robots, but rather by modifying the recipe itself. One convenient scheme to modify the recipe is to introduce deliberate delays, hereafter called queues q_j, to the non-critical process steps in order to achieve timing which resolves conflicts without compromising the throughput or on-wafer results delivered by the tool. Such a scheme, used in conjunction with Equation (1), is the basis for the "synchronous algorithm." To recap, a recipe as originally prescribed may introduce competing service requests which result in delays at critical process and gating steps, thereby degrading the wafer quality and throughput of the single-wafer cluster tool. The aim of "synchronous algorithm" is to insert intentional delays at non-critical process steps in order to ensure that no delays occur at critical process steps or gating steps, and thereby ensure guarantees of throughput and wafer quality.

Solving For The Queues

We shall now demonstrate how to solve for the delays, or q_j. Let τ be the timing of robot tasks as dictated by a prescribed recipe per Equation (1). By adding queues q_j to the process time p_j to modify the recipe, the new timing τ_i* is given by:

$$\tau_i^* = \sum_{j=1}^{i}(p_j + q_j) + \sum_{j=1}^{i-1} t_j - INT\left(\sum_{j=1}^{i}(p_j + q_j) + \sum_{j=1}^{i-1} t_j\right)$$

$$= \tau_i + \sum_{j=1}^{i} q_j - INT\left(\sum_{j=1}^{i}(p_j + q_j) + \sum_{j=1}^{i-1} t_j\right) + INT\left(\sum_{j=1}^{i} p_j + \sum_{j=1}^{i-1} t_j\right) \qquad (2)$$

The objective is to find a set of queues q_i to be inserted at the non-critical process steps such that the time interval between any two modules k and m, k=1, 2, 3, ...N; and

m = k, k+1,...,N, where module k and module m are assigned to have their wafers picked up by the same robot, is greater than the transport time of the robot. This would allow for time intervals sufficient for the robot to service all modules and thus avoid having to serve more than one module at a given time. However, the queues so derived must also be small enough to avoid excessive idling of modules. And there should be no queues at critical process modules or the gating module.

The set of queues will be solved for using Equation (2). This yields a system of linear equations $(\tau_i^* - \tau_i) = \sum_{i=1}^{N} a_{ij} q_j$; where a_{ij} is a lower triangular matrix with $a_{ij} = 0$, for $i < j$; and $a_{ij} = 1$, for $i \geq j$:

$$\begin{bmatrix} \tau_1^* - \tau_1 \\ \tau_2^* - \tau_2 \\ \tau_3^* - \tau_3 \\ \tau_4^* - \tau_4 \\ \bullet \\ \bullet \\ \tau_{N-1}^* - \tau_{N-1} \\ \tau_N^* - \tau_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \bullet \\ \bullet \\ q_{N-1} \\ q_N \end{bmatrix} \quad (3)$$

The constraint that no delays should occur at critical modules is now applied to Equation (3). For example if module #3 and #4 are critical, Equation (3) should be modified to the linear equations as shown below.

$$\begin{bmatrix} \tau_1^* - \tau_1 \\ \tau_2^* - \tau_2 \\ 0 \\ 0 \\ \bullet \\ \bullet \\ \tau_{N-1}^* - \tau_{N-1} \\ \tau_N^* - \tau_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \bullet & \bullet & 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \bullet \\ \bullet \\ q_{N-1} \\ q_N \end{bmatrix} \quad (4)$$

In Equation (4) above, the timings τ_i as prescribed by the original recipe are known. The target timings τ_i * are set to values which eliminate conflicts between all modules using the same robot, as described earlier. Thus LHS of Equation (4) are known values. The vector q_i is then solved for by pre-multiplying $(\tau_i^* - \tau_i)$ with the inverse of the modified constraint matrix as shown in Equation (5) below. Adding this set of q_i to the corresponding module process time p_i will synchronize wafer transport with wafer process.

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \bullet \\ \bullet \\ q_{N-1} \\ q_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \bullet & \bullet & 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & 0 & 0 & \bullet & \bullet & \bullet & \bullet \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} \tau_1^* - \tau_1 \\ \tau_2^* - \tau_2 \\ 0 \\ 0 \\ \bullet \\ \bullet \\ \tau_{N-1}^* - \tau_{N-1} \\ \tau_N^* - \tau_N \end{bmatrix} \qquad (5)$$

An Application of the Synchronous Algorithm

Specific embodiments of the invention will now be further described by the following, nonlimiting example which will serve to illustrate in some detail various features of significance. The example is intended merely to facilitate an understanding of ways in which the invention may be practiced and to further enable those of skill in the art to practice the invention. Accordingly, the example should not be construed as limiting the scope of the invention.

The synchronization of wafer transport with wafer process in a cluster tool will now be illustrated using a specific example of a cluster tool. Figure 3 is a schematic for a conceptual plan view of a wafer processing apparatus. The resist coating and developing modules are identified as CT 300 and DEV 302 respectively. Also shown in Figure 3 are the different bake modules vapor prime (VP) 304, soft bake (SB) 306, post exposure bake (PEB) 308, and hard bake (HB) 310, as well as their corresponding chill modules. The arrows connecting adjacent bake and chill modules represent the inter bay transfer arms, IBTAs 312, that transfer the substrate between these modules. As a result, the locations of these bake modules constrain the location of their

corresponding chill plate. The cassette end station (CES) robot 314 shown in the figure transfers substrates from and to the cassette end station. The stepper interface (SI) robot 316 transfers substrate from and to the stepper interface. The I/O module 318 is a buffer zone for the substrate transported to the stepper interface if and when it becomes necessary. The main robot 320 is the means to transport the substrate between all other modules such as vapor prime chill (VPC) to resist coat (CT.)

Figure 4 is a schematic for the wafer process flow. As can be seen from the schematic, when the transport means is the IBTA the substrate will have only one option in the flow diagram. This is the case when the substrate is transported from a vapor prime bake 400 to vapor prime chill 402. However, when the transport means is the main robot the substrate could have several options. For example, when a substrate is removed from a resist coat module 404, it can be transported to any of the soft bake modules 406 shown in Figure 4.

The Synchronization Algorithm

We will now demonstrate the application of the synchronization algorithm to this cluster tool as a series of four steps:

- **Step 1 *Input* The Recipe And Throughput Requirement**. This discussion refers to Figure 5. This step commences by inserting the recipe in a table 500. The first two columns 502 list the process steps in sequential order. The sending period, also referred to as the "system takt time" is also noted 504. Cycle times, also known as module takt times, are then calculated for each module in the "Module type" column 502 to ensure each module takt time is less than the system takt time. The module takt time for each process step is listed in a column 506. If not, redundant modules are added to reduce module takt time. As will be clear to one skilled in the art, for each process step:

  No. Modules Required = INT (Module takt time / System takt time)

In this example, most modules require one additional redundant module. The number of modules required for each process step is listed in a column 508.

- **Step 2 Determine The *Load* of The Wafer Transport**. As defined earlier, the "load" of a robot refers to the number of moves it is scheduled to make as well as the times allotted for the robot to perform these moves, as measured from the beginning of the current sending period. The determination of the robot load is depicted in the table 600 in Figure 6. This cluster apparatus of this example has twelve process steps. The timings of the twelve corresponding robot moves are determined as follows. Counting the time from the moment the wafer leaves the load port (cassette), cumulate the process times of each module and transport times up to the module of interest. For example, for the module code-named VP 602 (hereafter all modules are coded), it is (62+6) = 68 sec; for module VPC 604, it is (68+65+5) = 138 sec. The actual timings in seconds for all robot moves are listed in a column marked "Actual" 606. To determine the relative pick-up times T_i, the actual times are divided by the actual sending period. For example, dividing the actual pick-up times of VP and VPC by the actual sending period of 45 sec results, respectively, in normalized pick-up times 1.511 and 3.067. The normalized pick-up times for each of the twelve process steps are listed under the column "T_i Normalized" 608.

  Subtracting the integer portion of the T_i values results in the τ_i values, i.e., the available time the respective modules have to be served by robots, as measured from the beginning of sending period, and expressed in units of one sending period. To illustrate, the τ_i values of VP and VPC are 0.511 and 0.067, which indicates that VP must be serviced within .511 intervals of one sending period, and VPC must be serviced within .067 intervals of one sending period. The list of normalized τ_i values are listed in a column 610. The diagram 612 is a pictorial representation of the load: a total of twelve moves and the time they have to be served within a sending period. Figure 7 shows the

same information from another perspective. Since the robot transport times are around 5 & 6 seconds, the number of moves one robot can make within a sending period of 45 sec is (45/6)~7; say 6 moves to be on the conservative side. Thus 6 vertical lines are drawn in the graph 700. When the times the modules have to be served by a robot fall within one interval, such as SBC 702, DEV 704, HB 706, and Cassette 708, all of which fall into one interval 710, it means that they are competing for the same move of the robot at a given time. To elaborate, if there are two or more tasks which use one robot, and if the τ values of these tasks fall into one of the 6 intervals, it means that there is insufficient time for the robot to service each of the two or more robots.

These "conflicts" between the tasks for use of the robots are resolved as described in the subsequent steps.

- **Step 3 *Allocate* The Transport Load**. The first step to resolve conflicts in the manufacturing system is to allocate the load equally among robots in order to achieve a balanced transport load. Although conflicts are recipe dependent, assigning fewer loads per robot still reduces the chance for conflict. However, the possibility of balancing the load is dependent on the layout of the modules relative to robots. Poor layout limits accessibility to modules by robots and makes balanced loads difficult to attain. In this example, the layout is such that two robots, CES and SI, can each only serve two modules, leaving the bulk of the load to the main robot C-1 and three dedicated robots known as IBTA (Inter Bay Transfer Arm). The best allocation of transport load, under the constraints of the layout, is as shown in Figure 7. Six modules, VPC 712, CT 714, SBC 716, PEBC 718, DEV 720, and HB 722 are allocated to the main robot C-1 with three of them, SBC, DEV and HB competing for one move of the robot, as discussed earlier. These conflicts will be resolved by queuing as demonstrated in the next step.

- **Step 4 *Queuing* For Synchronization**. The information in the table 800 of Figure 8 is a summary of load allocation from Step 3. Only the

six modules served by main robot C-1, i.e., VPC 802, CT 804, SBC 806, PEBC 808, DEV 810, and HB 812, need to be considered for queuing. The remaining modules should not experience conflicts since each has dedicated robot, i.e., an IBTA, serving them. In the Target Column 814, the target timing profile is set for the 6 modules. For each of the modules listed in the table, a corresponding value is set for τ^*, where τ^* is an updated value for τ which eliminates conflicts between tasks for robots. Since only 3 of these 6 modules, SBC 806, DEV 810, and HB 812 are in conflict, only two, DEV and HB, need to have timing targets different from the original values for τ prescribed by the recipe. The timing targets are listed in a column 814, and are set such that the timing interval between any pair of the 6 modules is larger than robot transport time (=6/45~0.1333). The differences between the target and the originally prescribed timing profiles are referred to as the gaps and are computed the column 816. These are shown pictorially in the graph 818 adjacent to the table. Another objective of the "synchronous algorithm" is to ensure that no delays are introduced at critical process steps. In this example, critical process steps are step 3 804, step 4 806, and step 7 809. No queues should be added to modules corresponding to these steps, i.e., the target timing for these modules should be the same as the prescribed values. The gaps computed in the Gap column 816 may now be substituted in to Equation (3) to solve for the queues that will close these gaps. However, to ensure zero delays at the critical process modules, the matrix 818 relating gaps and queues must be modified per Equation (4) to generate a modified matrix 820. Pre-multiplying the gaps from the Gap column 816 with the inverse of the modified matrix 822 produces the queues needed to close the gap 824. The solution for the queues is transferred to a Que Column 826. The solution, which is in units of the sending period, is converted to actual time in an Actual Que Column 828.

- **Step 5 *Check* The Solution**. The queues determined in Step 4 are now added to the module processing time of the original recipe. This is to

verify if conflicts have been resolved. This is in fact the case as shown in the figure.

Robot Assignment

Another aspect of the scheduling problem, which merits automation, is the assignment of robots to modules. For example, in step three listed above, a recipe was chosen which assigned a single robot between each pair of consecutive models; this allocation is shown in the recipe listed in Figure 7. The allocation was chosen amongst many possible allocations.

In general, there is a need for an algorithm that determines an optimal robot assignment prior to the determination of queues. The need for such an algorithm will be demonstrated in the following example. Suppose we have a simplified track system, consisting of three modules, labeled Mod1, Mod2 and Mod3. Suppose we have two robots, Robot1 and Robot2, both of which can service all three modules. Let the sending period be designated by the variable SP. Suppose τ_1=0.0, τ_2=0.6, τ_3=0.7, in units of the sending period, and suppose the robots can move in 0.3, in units of SP. There are four possible robot allocations:

Assignment	Mod1 to Mod 2	Mod2 to Mod3
1.	Robot1	Robot1
2.	Robot2	Robot2
3.	Robot1	Robot2
4.	Robot2	Robot1

Upon inspection, only assignments 3 and 4 are viable. In assignments 1 and 2, the time interval between τ_2=0.6 and τ_3=0.7 is 0.1 sending periods, which is less than the 0.3 sending periods required for a robot to move. Hence the optimal allocations are, in this case, assignment #3 and #4; as the time intervals between the values have sufficient distance, this robot assignment obviates the need to insert delays. Other criteria may also enter into the determination of an optimal robot assignment, for instance, balancing loads, increasing throughput. An algorithm is necessary which would determine, in cases more complicated than the simplified example above, an optimal robot assignment.

One method of performing such an assignment is simply an exhaustive technique: generate all possible robot assignments, and determine the validity of each assignment, i.e., ensure that all modules assigned to a robot differ sufficiently in their τ values to permit the robot to service them. The assignments thus generated may also be selected on additional criteria, such as load balancing.

Solving for Updated Timings (τ^*)

Another feature of the synchronization that merits automation is the derivation of the updated timings, given by τ^*. To elaborate, in step 4 of the algorithm outlined earlier, the algorithm was fed updated values of τ^*, where for any two modules which shared a robot, the respective τ^* values differed by enough time to allow the robot to move between them. There is a need for an automated method of deriving these τ^* values. One such technique is as follows:

For each robot with a conflict, take the τ value for each of its modules.

For each combination of these τ values, sort the τ values from lowest to highest. For each sorted list of τ values:

- Proceed sequentially through the τ values, from lowest to highest.
- Determine the difference between the given τ value and the one that precedes it
- If the difference is less than the time allotted for the robot to move, increment the τ value sufficiently.
- Go to next τ value.

When this algorithm finds an updated set of τ values for a given robot that eliminates conflicts, these become τ^* values. It can be proven that if a conflict-free group of τ^* values exist, the algorithm outlined above will find it.

Genetic Algorithms

The synchronization, robot assignment, and derivation problems can also be solved by use of a genetic algorithm (GA). A GA is an iterative process

that starts out with an initial population of genes, which encode possible states of the problem. This population is systematically improved with each iteration, through a process of selective breeding. To perform selective breeding, the GA needs to a) define the characteristics of a species, and b) judge a species' fitness.

Characteristics of a Species A specie is characterized by n genes. For our problem, we use two types of genes, one to represent the robot assignment, and the other, a queue segment. Consider the example used in the description of the earlier algorithm. The robot assignment will range from 1 to 4, indicating which robot will work on a particular module. Queue segment is also an integer indicating how many "time zones", i.e., robot move periods, are to be added to a module's arrival time in order to avoid conflict in robot assignment. In our earlier example, the modules arrive in six different time zones, as shown in figure 6 612. If a time zone sees the arrival of, for instance, five modules, a conflict results. The addition of a queue segment to one of the modules will push the arrival time to the next time zone and hence resolve the conflict.

Fitness of a Species We can measure fitness by the reciprocal of a species 'badness'. In turn, badness can be measured by a weighted sum of the degree of conflict and the number of added queue segments. An ideal species is one that has no added segments, and results in no conflict in robot assignment.

To derive the fitness function, we scan each time zone and count the number of redundant assignments for each robot. The results are summed for all robots and all time zones. Call this sum *s*. We proceed to count the number of added queue segments, and call it *t*. The fitness function is then

$$f(s,t) = \frac{1}{(1 + w_1 s + w_2 t)};$$

where the weights w_1 and w_2 are assigned according to the relative importance of *s* over *t*.

Recipe Cascading

An embodiment of the invention allows the cluster tool to change from a first recipe to a second recipe, while preserving periodicity, and ensuring that there are no delays at critical points. This procedure is referred to as recipe cascading. Cascading involves emptying a first lot of wafers, hereafter called the *exiting lot*, off a cluster tool, and simultaneously and serially populating the cluster tool with another lot of wafers, hereafter called the entering lot. The procedure will be performed with no delays incurred at critical process steps; and with no additional robots and process modules other than those called for by the recipe and throughput requirements of the entering and the exiting lots. The entering lot may also have different recipe and throughput requirements from the exiting lot.

Notation

The following parameters will be employed in our discussion of recipe cascading:

m^{ex} = number of process steps, superscript (en, ex) hereafter denotes entering or exiting lot.

n^{ex} = number of exiting wafers to fully empty, or of enter wafers to fully populate the cluster tool.

SP_i^{ex} = Send period of the *i*th wafer which is the time interval between the launching of the

*(i-1)*th and *i*th wafer.

$SP_\bullet^{ex}$ = Send period of the wafers during steady state.

$SP_{1st,j}^{en}$ = The delay in the launching of the 1st wafer of the entering lot to accommodate the change-over at the *j*th process module of the exiting lot.

CO_j = Time it takes to change the *j*th process module of the exiting lot over to a new setting.

T_{ijk}^{in} = The arrival time of the *i*th wafer at the *j*th module transported by the *k*th robot.

T_{ijk}^{out} = The departure time of the *i*th wafer from the *j*th module transported by the *k*th robot

p_j^{ex} = Process time of *j*th process step of the exiting lot.

t_j^{ex} = Transport time between *(j-1)*th and *j*th process step of the exiting lot.

q_{ij}^{ex} = queues added to the *i*th wafer at the *j*th process step of the exiting lot.

Nature of Recipe Cascading

Figure 9 illustrates a typical recipe 900. It specifies the process and transport tasks and the timing of each task to be performed on a wafer as the wafer goes through a cluster tool. To satisfy throughput requirement, succeeding wafers are sent through the cluster tool at a constant send period, as illustrated in Figure 10. This send period SP 1000 is given by

$$\mathrm{SP} = \frac{3600}{\mathrm{WPH}}$$

where WPH is the throughput requirement in wafer per hour.

By the *n*th wafer, the cluster tool will be fully populated with wafers. For every wafer exiting the cluster tool, there is a wafer entering to replenish it. All process and transport tasks performed on the wafers occur in a periodic fashion, the periodicity being delimited by the send period. When the system reaches this stage, it is said to be in steady state. The number of wafers *n* required to fully populate the cluster tool and ramp up to a steady periodic state is given by the expression

$$n = 1 + \mathrm{INT}\left(\frac{\pi}{\mathrm{SP}}\right); \qquad \text{(A)}$$

where π is the total process and transport time performed on a wafer as indicated in Figure 9 902. The symbol $\mathrm{INT}(\bullet)$ denotes a function that rounds a number down to the nearest integer. The above equation also applies for the number of wafers for a lot to ramp down from a steady state and fully empty out the cluster tool. In the steady state, every wafer has identical process and transport tasks performed on it at identical time intervals. There is therefore no need to keep track of the movement of wafers inside the cluster tool.

During the recipe cascading, however, the wafers are in transition from the steady state of the exiting lot to the steady state of the entering lot, as illustrated in Figure 11. During the transition, some wafers will be processed and transported according to the recipe and send period prescribed for the exiting lot; others, will be processed and transported according to the recipe prescribed for the entering lot. Periodicity therefore can not be maintained. "Conflicts" will occur. "Conflicts" refer to two situations. In one situation called process conflict, two wafers have to be processed sequentially by the same process module within an interval shorter than the process time of that step. In the other situation called transport conflict, two wafers have to be transported sequentially by the same robot within an interval shorter than the robot transport time.

One solution to conflicts is to add more modules and robots. This solution is costly and not practical. Another solution is to devise a set of priority rule and implement "if-then" algorithm to react to the conflicts. Because of its combinatorial nature, this solution generates huge combination of outcomes that eventually lead to chaos and unpredictability. The fundamental solution is to eliminate conflicts completely through proper timing in launching wafers of the entering lot and insertion of intentional delays at the non-critical process steps of both the exiting and entering lot. This is the principle behind recipe cascading described herein.

Procedure

Step 1: Estimate the number of wafers in transition. During transition, the exiting lot is emptying while the entering lot is populating the cluster tool. The number of wafers involved in the transition can be estimated by using Equation (A). The algorithm for the estimation is as follows.

$\pi^{ex} = 0$

For i = 1, 2... m^{ex}

$\pi^{ex} = \pi^{ex} + p_i^{ex} + t_i^{ex}$

Next i

$n^{ex} = \mathrm{INT}\left(\pi^{ex} / SP_{\bullet}^{ex}\right)$

$\pi^{en} = 0$

For i = 1, 2... n^{en}

$\pi^{en} = \pi^{en} + p_i^{en} + t_i^{en}$

Next i

$n^{en} = INT(\pi^{en} / SP_{\bullet}^{en})$

Step 2: Estimate the lower bound of "delay". The timing of launching the wafers of the entering lot is a primary variable responsible for conflict occurrence. Therefore proper timing of the launches is important. Additionally, the timing of launching the first wafer, hereafter called the "delay", should be such that it provides the time needed to change a process module over to a new setting called for by the entering lot. So that when the first wafer of the entering lot arrives, the change over is complete and the process module is ready to accept it. There will be as many "delays" as the number of module change over called for by the recipe of the entering lot. The maximum of them is the lower bound of the delay. Any amount of delay contemplated for conflict resolution must be greater than the lower bound. Referring to Figure 12, the lower bound can be derived as follows.

$$SP_{1st,j}^{en} + \sum_{l=1}^{j-1}\left(p_l^{en} + t_l^{en}\right) = CO_j + \sum_{l=1}^{j}\left(p_l^{ex} + t_l^{ex}\right)$$

$$SP_{1st,j}^{en} = CO_j + \sum_{l=1}^{j}\left(p_l^{ex} + t_l^{ex}\right) - \sum_{l=1}^{j-1}\left(p_l^{en} + t_l^{en}\right)$$

$$SP_{1st,\bullet}^{en} = max\left(SP_{1st,1}^{en}, SP_{1st,2}^{en}, SP_{1st,3}^{en}, \ldots SP_{1st,j}^{en}, j = 1,2,\ldots m^{en},\right) \qquad \text{(B)}$$

The following is the algorithm for estimating the lower bound of delay per Equation (B)

For j =1, 2... m^{en}

$SP_{1st,j}^{en} = CO_j$

For l =1, 2...j

$SP_{1st,j}^{en} = SP_{1st,j}^{en} + \left(p_l^{ex} + t_l^{ex}\right) - \left(p_l^{en} + t_l^{en}\right)$

next l

$$SP^{en}_{1st,j} = SP^{en}_{1st,j} - \left(p^{en}_{j} + t^{en}_{j}\right)$$

next j

$$SP^{en}_{1st,\bullet} = max\left(SP^{en}_{1st,1}, SP^{en}_{1st,2}, SP^{en}_{1st,3}, \ldots SP^{en}_{1st,j}, j = 1,2,\ldots m^{en},\right)$$

Step 3: Track the movement of wafers in transition. The movement of wafers in transition can be tracked. The arrival time T^{in}_{ijk} of the *i*th wafer at the *j*th module transported by the *k*th robot is the departure time $T^{out}_{i(j-1)k}$ from the prior *(j-1)*th process module plus the transport time t_j from the *(j-1)*th to the *j*th module:

$$T^{in}_{ijk} = T^{out}_{i(j-1)k} + t_j \qquad \text{(C)}$$

In turn, the departure time from the *j*th module is the arrival time plus the processing time and queues deliberately inserted for conflict resolution:

$$T^{out}_{ijk} = T^{in}_{ijk} + p_j + q_{ij} \qquad \text{(D)}$$

The recursive formulae of Equations (C) and (D) above permit one to map out the movement of the wafers in transition. Thus, measuring time from when the last wafer of the exiting lot is launched and transition begun, see Figure 11, the arrival and departure of a wafer at a process module can be calculated per the algorithm below. The arrival and departure times outside of the transition are not considered and therefore marked with large negative values.

For wafers of the exiting lot, see Figure 11:

For i =1, 2... n^{ex}

$T^{in}_{i1k} = -\infty$

Sum = 0

For j =1, 2... m^{ex}

Sum = Sum + $p^{ex}_{j} + q^{ex}_{ij}$ (E)

T^{out}_{ijk} = Sum – $\left(n^{ex} - i\right) * SP^{ex}_{\bullet}$

if j = m^{ex}, go to out

$T^{in}_{i(j+1)k} = T^{out}_{ijk} + t^{ex}_{j}$

if $\left(T^{in}_{i(j+1)k} \le 0\ , -\infty\ , T^{in}_{i(j+1)k}\right)$

if $\left(T^{out}_{ijk} \le 0\ , -\infty\ , T^{out}_{ijk}\right)$

out: next j

next i

For wafers of the entering lot,

Sum = 0

For i = $\left(n^{ex}+1\right), \left(n^{ex}+2\right) \ldots \left(n^{ex}-1+n^{en}\right),\ \left(n^{ex}+n^{en}\right)$

Sum = Sum + $SP^{en}_{i-n^{ex}}$ (Note: $SP^{en}_{I} \equiv$ delay) (F)

Psum = Sum

For j =1, 2... m^{en}

Psum = Psum + $p^{en}_{j} + q^{en}_{ij} + t^{en}_{j}$ (G)

T^{out}_{ijk} = Psum - t^{en}_{j}

if j = m^{en}, go to out1

$T^{in}_{i(j+1)k} = T^{out}_{ijk} + t^{en}_{j}$

Out1: next j

next i

Step 4: Identify conflicts. Having determined the arrival time T^{in}_{ijk} and the departure time T^{out}_{ijk}, one can check for conflicts. A process conflict occurs when a process module at the *j*th process step is called on to process sequentially a pair of wafer, the *m*th and the *n*th, within a time interval shorter than the process time. In other words, the *n*th wafer arrives before the *m*th wafer departs from the module. Thus conflict occurs if the following is true:

$$\mathrm{OR}\left\{\mathrm{AND}\left[\left(T^{out}_{mjk} \ge T^{out}_{njk}\right), \left(T^{out}_{njk} \ge T^{in}_{mjk}\right)\right], \mathrm{AND}\left[\left(T^{out}_{mjk} \le T^{out}_{njk}\right), \left(T^{out}_{mjk} \ge T^{in}_{njk}\right)\right]\right\}$$

By using above logic statement to examine all possible combination of (m,n) pairs for all process steps, all process conflicts may be identified:

For j = 1 to $\left(m^{ex}+m^{en}\right)$

For m = 1 to $(n^{ex}+n^{en})$

For n = 1 to m-1

Conflict if

$$OR\left\{AND\left[\left(T_{mjk}^{out}\geq T_{njk}^{out}\right),\left(T_{njk}^{out}\geq T_{mjk}^{in}\right)\right],AND\left[\left(T_{mjk}^{out}\leq T_{njk}^{out}\right),\left(T_{mjk}^{out}\geq T_{njk}^{in}\right)\right]\right\}=\text{TRUE}$$

next m

next n

Similarly, a transport conflict occurs when a robot, the kth robot, is called on to transport sequentially two wafers, the mth and the nth, within a time interval shorter than the transport time. In other words, the time interval between the arrival of the mth wafer and the departure of the nth wafer is shorter than the transport time. Thus transport conflict occurs if the following is true:

$$OR\left[\left|T_{mjk}^{out}-T_{njk}^{out}\right|<g,\left|T_{mjk}^{out}-T_{njk}^{in}\right|<g,\left|T_{mjk}^{in}-T_{njk}^{out}\right|<g,\left|T_{mjk}^{in}-T_{njk}^{in}\right|<g\right]$$

In the above, the symbol $|\bullet|$ denotes absolute value and "g" is the time allocated for the robot to make one transport move. The time "g" is greater or equal to the transport time of the robot. By using above logic statement to examine all possible combination of (m,n) pairs for all process steps, all transport conflicts may be determined:

For j = 1 to $\left(m^{ex}+m^{en}\right)$

For m = 1 to $(n^{ex}+n^{en})$

For n = 1 to m-1

Conflict if

$$OR\left[\left|T_{mjk}^{out}-T_{njk}^{out}\right|<g,\left|T_{mjk}^{out}-T_{njk}^{in}\right|<g,\left|T_{mjk}^{in}-T_{njk}^{out}\right|<g,\left|T_{mjk}^{in}-T_{njk}^{in}\right|<g\right]=\text{TRUE}$$

next m

next n

Step 5: Resolve conflict through proper queuing and launching. Note that in Equations (E) through (G), there are three yet undetermined variables used in the calculation of arrival and departure time of a wafer at a process module by a particular robot. These are the intentional delays called queues q_{ij}^{ex}, q_{ij}^{en} and the timing of launching the wafer of the entering lot SP_i^{en}. Optimization procedures are used to find the best combination of these three variables such that there is no conflict and the total queues is minimized. Genetic Algorithms are one such optimization procedure. Others will be apparent to one skilled in the art.

The foregoing description of various embodiments of the invention has been presented for purposes of illustration and description. It is not intended to limit the invention to the precise forms disclosed. Many modifications and equivalent arrangements will be apparent.

CLAIMS

What is claimed is:

1. A method for wafer processing comprising:

loading a first plurality of wafers into a wafer cluster tool, wherein the first plurality of wafers are loaded into the cluster tool individually at intervals delimited by a first sending period, and each wafer of the first plurality of wafers is processed according to a first recipe, the first recipe including

a first sequence of process steps

a first plurality of process times, wherein each process step in the first sequence of process steps has a corresponding process time in the first plurality of process times;

loading a second plurality of wafers into a wafer cluster tool, wherein the second plurality of wafers are loaded into the cluster tool individually at intervals delimited by a second sending period, and each wafer in the second plurality of wafers is processed according to a second recipe, the second recipe including

a second sequence of process steps;

a second plurality of process times, wherein each process step in the second sequence of process steps has a corresponding process time in the second plurality of process times;

wherein the cluster tool has a transition period, during which the cluster tool processes one or more wafers from the first plurality of wafers according to the first recipe and one or more wafers from the second plurality of wafers according to the second recipe.

2. The method of claim 1, wherein the first and second sequences of process steps are identical.

3. The method of claim 2, wherein the first sending period and the second sending period are equal.

4. The method of claim 2, wherein the first and second plurality of wafers are processed in a wafer cluster tool.

5. The method of claim 4, wherein the wafer cluster tool includes a plurality of processing chambers.

6. The method of claim 5, wherein each process step from the first sequence of process steps is performed in a processing chamber from the plurality of processing chambers.

7. The method of claim 6, wherein each wafer of the first plurality of wafers is released from the wafer cluster tool at intervals delimited by the first sending period.

8. The method of claim 7, wherein each wafer of the second plurality of wafers is released from the wafer cluster tool at intervals delimited by a second sending period.

9. A computer program product for scheduling a wafer processing system, the computer program product comprising:

resources for scheduling the wafer processing system during a first time period, wherein a first plurality of wafers is processed during the first time period, and each of the first plurality of wafers is processed according to a first recipe;

resources for scheduling the wafer processing system during a second time period, wherein a second plurality of wafers is processed during the second time period, and each of the second plurality of wafers is processed according to a second recipe;

resources for scheduling the wafer processing system during a third time period, wherein a third plurality of wafers is processed during the third time period, such that one or more wafers from the third plurality are processed according to the first recipe, and one or more wafers from the third plurality are processed according to the second recipe

10. The computer program product of claim 9, wherein the computer program product resides on a server coupled to the wafer processing system.

11. The computer program product of claim 10, wherein the wafer processing system comprises a wafer cluster tool.

12. The computer program product of claim 11, wherein the wafer cluster tool includes a plurality of robots for transferring the first, second, and third plurality of wafers.

13. The computer program product of claim 12, wherein the wafer cluster tool includes a plurality of process chambers for processing the first, second, and third plurality of wafers.

14. The computer program product of claim 13, wherein the computer program product schedules the wafer cluster tool to eliminate conflicts amongst the plurality of process chambers for use of the plurality of robots.

15. The computer program product of claim 11, wherein the computer program product schedules the wafer processing system in real-time during the first, second, and third time periods.

16. The computer program product of claim 10, wherein the computer program product uses a genetic algorithm to schedule the wafer processing system.

17. The computer program product of claim 10, wherein the computer program product uses a linear transformation to schedule the wafer processing system.

18. The computer program product of claim 10, wherein the computer program product uses a first linear transformation to schedule the wafer

processing system during the first time period, uses a second linear transformation to schedule the wafer processing system during the second time period, and uses a genetic algorithm to schedule the wafer processing system during the third time period.

19. The computer program product of claim 10, wherein the first plurality of wafers are processed at a first constant rate.

20. The computer program product of claim 19 wherein the second plurality of wafers are processed at a second constant rate.

21. A method of operating a wafer cluster tool, wherein the wafer cluster tool includes a plurality of process chambers for processing a plurality of wafers, the method comprising:

loading the plurality of wafers into the wafer cluster tool, wherein each wafer of the plurality of wafers is loaded after an end of one sending period, the sending period comprising a uniform time interval;

processing the plurality of wafers in the cluster tool, wherein the plurality of wafers are processed concurrently in the cluster tool according to a plurality of recipes, wherein each recipe of the plurality of recipes includes a plurality of process times for the process chambers in the wafer cluster tool.

22. The method of claim 21, wherein the plurality of wafers includes two or more wafers which are processed according to a first recipe in the plurality of recipes, and two or more wafers which are processed according to a second recipe in the plurality of recipes.

23. The method of claim 22, wherein the wafer cluster tool includes a plurality of robots for transferring the wafers amongst the plurality of process chambers in the wafer cluster tool.

24. The method of claim 23, wherein a number of robots in the plurality of robots is less than a number of process chambers in the plurality of process chambers.

25. The method of claim 24, further comprising:

releasing the plurality of wafers from the wafer cluster tool, wherein each wafer of the plurality of wafers is released from the cluster tool at the end of one sending period.

26. A method of scheduling a wafer cluster tool, wherein the wafer cluster tool is coupled to a computer system for scheduling the cluster tool, the method comprising:

inputting a plurality of recipes into the computer system, wherein each recipe of the plurality of recipes includes a plurality of process times for the cluster tool;

outputting a schedule for the wafer cluster tool, such that the schedule enables the wafer cluster tool to process a plurality of wafers concurrently using the plurality of recipes, wherein each wafer of the plurality of wafers is processed at a uniform rate.

27. The method of claim 26, wherein the schedule includes a plurality of pick-up times for a plurality of robots in the cluster tool

28. The method of claim 2, wherein the plurality of pick-up times eliminate conflicts for the plurality of robots.

29. The method of claim 26, wherein the schedule is computed by the computer system while the plurality of wafers is being processed.

30. The method of claim 28, wherein a genetic algorithm is used to determine the schedule.

31. The method of claim 28, wherein a linear transformation is used to determine the schedule.

32. The method of claim 28, wherein a linear transformation and a genetic algorithm are used to determine the schedule.

33. A wafer cluster tool comprising:

a plurality of processing chambers for processing a plurality of wafers, wherein the plurality of wafers are processed according to a plurality of recipes, such that each recipe of the plurality of recipes specifies a plurality of process times for a wafer from the plurality of wafers;

a plurality of wafer transporters for transporting the plurality of wafers between the process chambers;

a scheduler coupled to the wafer cluster tool, the scheduler including a computer program running on a computer system, wherein the scheduler ensures that the cluster tool processes the plurality of wafers concurrently.

34. The wafer cluster tool of claim 33, wherein the scheduler determines a schedule for the cluster tool by use of a genetic algorithm encoded in the computer program.

35. The wafer cluster tool of claim 33, wherein the scheduler determines a schedule for the cluster tool by use of a linear transform encoded in the computer program.

36. The wafer cluster tool of claim 33, wherein the scheduler determines a schedule for the cluster tool in real-time, while the plurality of wafers are processed by the cluster tool.

37. The wafer cluster tool of claim 33, wherein the wafer cluster tool processes each wafer of the plurality of wafers at a uniform rate.

38. The wafer cluster tool of claim 33, wherein at least one wafer transporter from the plurality of wafer transporters is used by two or more pairs of process chambers from the plurality of process chambers.