2023 24th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-3475-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISQED57927.2023.10129375

A Flexible Cluster Tool Simulation Framework with Wafer Batch Dispatching Time Recommendation

Hsin-Ping Yen [1], Shiuan-Hau Huang[1], Yan-Hsiu Liu[2], Kuang-Hsien Tseng[2], Ji-Fu Kung[2], Yi-Ting Li[1],
Yung-Chih Chen[3], and Chun-Yao Wang[1]

[1]National Tsing Hua University, Taiwan, ROC
[2]United Microelectronics Corporation, Taiwan, R.O.C.
[3]National Taiwan University of Science and Technology, Taiwan, R.O.C.

***Abstract*—The semiconductor manufacturing process consists of multiple steps and is usually time-consuming. Information like the turnaround time of a certain batch of wafers can be very useful for manufacturing engineers. A simulation model of manufacturing process can help predict the performance of manufacturing process efficiently, which is very beneficial to the manufacturing engineers. The simulation result can also deliver messages to system engineers for achieving better throughput after adjustment. In this work, we propose a flexible simulation framework for a cluster tool. We implemented the simulator in C++ language with SystemC. The batch information used for the design of simulator was gathered from industrial data. The experimental results show that there is only less than 2% difference between the simulation and the manufacturing data in terms of entire processing time, which indicates the high accuracy of the simulator. The experimental results with the proposed dispatching method achieve a higher throughput compared to the manufacturing data such that the dispatching time points can be recommended to the system engineers.**

***Index Terms*—cluster tool, simulation, wafer transferring system.**

I. Introduction

Semiconductor manufacturing process becomes more and more complex with the advances of wafer technology. Semiconductor manufacturing process includes etching, deposition, photo-lithography, etc. The system that handles the wafer transfer and processing is called a cluster tool. Due to the increasing number of processing steps, the cluster tools have also become more complicated. On top of the increasing number of processing steps, a cluster tool often processes multiple batches simultaneously for elevating the utilization of the buffer arm, which is the bottleneck of the system.

The throughput of a cluster tool for batches of wafers is valuable information to engineers. This information could be gathered from the manufacturing data. However, since the wafer manufacturing process is time-consuming, it is inefficient to obtain this information after completing the entire manufacturing process. Thus, a simulator for the cluster tool, which is very efficient as compared with the actual manufacturing process, is desired, and the throughput of a cluster tool can be estimated accordingly. A simulator can also show the time points for improvement, at which the system is able to be adjusted to boost its productivity.

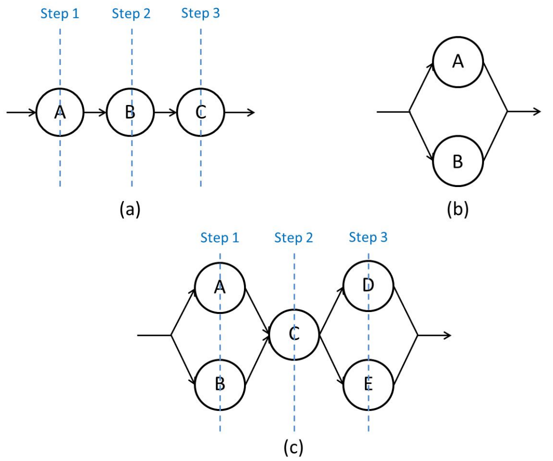


Fig. 1: Wafer paths of different cluster tool modes. (a) A wafer path under a serial mode: chamber A → chamber B → chamber C. (b) A wafer path under a parallel mode: (chamber A or chamber B). (c) A wafer path under a serial-parallel mode: (chamber A or chamber B) → chamber C → (chamber D or chamber E).

With the differences in objectives and manufacturers, cluster tools vary in their configurations, modes, and scheduling. The configuration of a cluster tool involves the number of chambers and load locks. Cluster tools can be classified by the number of arms for wafer transfer between chambers. Single-armed and dual-armed cluster tools are widely used. The modes in the cluster tools can be categorized into serial mode, parallel mode, or serial-parallel mode. Their wafer paths are shown in Fig. 1, where each node represents a chamber. A serial mode cluster tool sends each wafer to a series of chambers in a sequence. A parallel mode cluster tool sends each wafer to only one chamber among multiple available chambers. For a serial-parallel cluster tool, a wafer is sent to a series of chambers, but some steps may have multiple available chambers. This mode is often used when some steps have a longer processing time such that multiple chambers are reserved for those time-consuming steps.

Many simulation methods have been proposed over the

This work is supported in part by the Ministry of Science and Technology of Taiwan under MOST 109-2221-E-007-082-MY2, MOST 109-2221-E-155-047-MY2, MOST 110-2224-E-007-007, MOST 111-2218-E-007-010, MOST 111-2221-E-007-121, MOST 111-2221-E-011-137-MY3, and UMC: I-2020-10-184.

years [7] [8] [14] [17] [21], such as Cellular Automata [1] [18], event graph [13], and Petri Net [23], etc. The methods mentioned above are not easily portable when encountering different cluster tool configurations. [3] predicted the throughput of a cluster tool using machine learning methods. Machine learning methods can predict the throughput of the cluster tool, but it does not disclose the scheduling logic behind the system. There also have been some simulation platforms [22] developed for analyzing the wafer transfer procedure to increase the throughput. [22] provided the option to choose the mode between serial and parallel, but not the more complex serial-parallel mode. Furthermore, the simulation was only conducted between the chambers.

In this work, we address the portability issue of the cluster tools by separating the functionality of cluster tool components from the scheduling of the system. When adapting to a new configuration, the components in the cluster tools can be reused and the engineers just re-adjust the scheduling to match the new system. The modes in the cluster tools are different in their wafer paths. Our simulator uses the wafer path of the batch to determine the next location of the wafer such that different modes of the cluster tool can be accommodated.

Aside from the cluster tool configurations and modes, scheduling methods can also be different. The scheduling methods for cluster tools have been studied in the past [4]–[6], [9], [10], [19], [20], which directly influence the throughput of the cluster tool. In this work, we propose two scheduling methods. One is based on the state of the cluster tool, and the other is based on the sequence of arm movements.

Last, the throughput of the cluster tool can also be affected by the dispatching time points of batches. The dispatching method indicates the sequences and locations of the batches. If a batch of wafers arrives later, the idle time between batches is a waste. If a batch of wafers arrives earlier, the cluster tool might not have available resources for it to be processed. This batch is forced to wait at the cluster tool though it could be sent to another cluster tool for processing earlier. Batch dispatching is usually determined by experienced engineers, who are familiar with the cluster tools and control the timing and sequence of the batches to be processed. [15] [16] proposed dispatching rules regarding recipe arrangements with semiconductor fabrication plants (fabs) simulation, which simulate batches with different recipes being processed on a set of cluster tools in a factory. To prevent batches from arriving later or earlier, we propose a dispatching method, which recommends the dispatching time point to engineers. The experimental results also show that the simulations using the proposed dispatching method yield a higher throughput than the data from the fabs.

The main contributions of this work are as follows:

- We propose a flexible simulation framework, which can be deployed on a variety of cluster tools.
- We propose two scheduling methods that can be used in the simulator.
- The simulator using the proposed dispatching method recommends the dispatching time points for achieving a higher throughput.

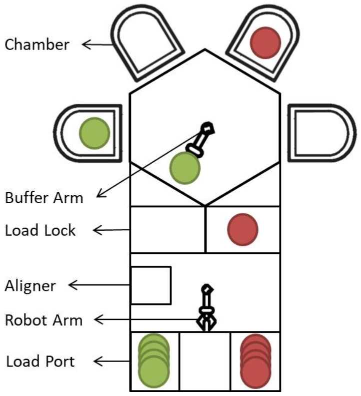


Fig. 2: Single cluster tool configuration.

II. Background

A. Cluster Tool Configuration

As shown in Fig. 2, a cluster tool has the following components: Load Port, Robot Arm, Aligner, Load Lock, Buffer Arm, and Chambers.

After a batch of wafers arrives at the load port, each wafer will be transferred to chambers sequentially. For instance, assume that the recipe is a single-step process, the wafer path is as follows: Load Port → Aligner → Load Lock → Chamber → Load Lock → Load Port.

We detail a wafer's journey inside a cluster tool using Fig. 2. After the wafer is transferred to the aligner by the robot arm, the aligner starts to rotate the wafer. We need to ensure that the wafer is perfectly aligned when being placed into the chamber. After the alignment, the wafer is transferred into the load lock. There is a gate on each side of a load lock. Both gates have to be closed for performing the air venting and pumping procedure. The wafer is currently on its way to the chamber. The load lock starts pumping air to raise the air pressure to the same level as it is on the chamber side. When the load lock opens the gate on the chamber side, the buffer arm then transfers the wafer into the appointed chamber. Each chamber has a gate, which is closed during processing. After being processed, the wafer travels back to the load lock. The load lock performs the air venting and cooling procedure simultaneously before opening its gate on the load port side. Finally, the wafer returns back to the load port, where the original batch arrived.

Note that a wafer will pass through a loading and unloading phase between two components. The loading and unloading duration can be changed depending on different types of cluster tools. For some cluster tools, a load lock is assigned to a specific batch of wafer. However, for the others, a load lock is shared among different batches. As a result, a simulation framework needs to accommodate those differences.

B. Inputs/Outputs and Features of the Simulator

The following items are the inputs of the simulator:

- Wafer batch information: Batch names, recipes, processing time, assigned load ports, and dispatching time point.

- Parameter settings: Time duration for transferring wafers using the robot arm and the buffer arm, wafer loading and unloading duration, pumping and venting duration at load locks, and aligning duration at the aligner.
- Buffer arm and robot arm's wafer transferring order: This is produced by different scheduling methods. More details will be explained in Section III-B.

The output of the simulator is a log file indicating the duration of each wafer at each component. It is presented in Gantt chart [2] such that it can be graphically analyzed by engineers.

The simulator has the following features:

- System Portability: The simulator can be easily deployed on different types of cluster tools.
- Scheduling diversity: The simulator accepts different kinds of scheduling methods for performance comparison under the same system configuration.
- Throughput estimation and dispatching time recommendation: After simulation, the dispatching time of batches is recommended for achieving a higher throughput.

C. Log File

In this work, we need two types of log files to construct the simulator. The first one records all the wafer movements in the cluster tool. It contains the details including wafer ID, lot ID, from/to locations, and slot number of load locks and load ports. This type of log file provides the configuration of cluster tool and the process recipe, which indicates the processing steps of each batch of wafers.

The other type of log file is an event log, which details the operation steps for each part of the cluster tool. For instance, the operation steps of load lock include the gate opening/closing, air pumping/venting, and cooling procedures. Those are events recorded in the event log, and we can examine the relationship between events to understand the logic of the entire system. More details will be discussed in Section III.

III. Proposed Framework

This section describes the proposed simulator framework. We also emphasize the design insights that allow the simulator to have the features mentioned in Section II-B. For the system portability, our design separates the scheduling methods from the functionality of each component. This is because when the components work independently, it is easy to reuse them for constructing a new simulator with different configurations. For the scheduling diversity, we propose two methods. The first one uses the state of the cluster tool to determine the corresponding action. The other one uses a *component string* to indicate the movement of the buffer arm and the robot arm. For batch dispatching, we propose a method that determines the input dispatching time points for each batch such that a higher throughput of a cluster tool is achieved. The simulator also provides the option to use the input dispatching time points given by engineers.

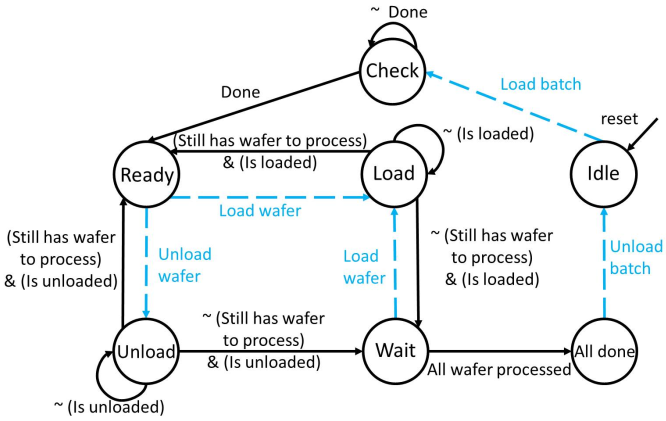


Fig. 3: The Finite State Machine for Load Port.

A. Module Construction

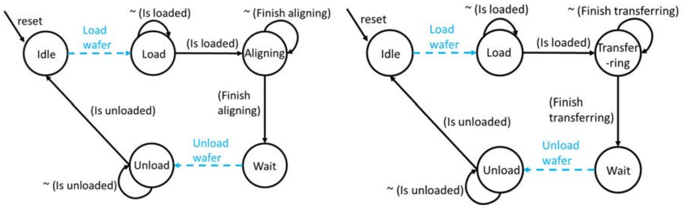
We model each component of the cluster tool by Finite State Machines (FSMs) [11] [12]. In addition to the existing components of the cluster tool, we also need a scheduling module and a dispatching module. The scheduling module decides the action of each component based on the state of the cluster tool. The dispatching module controls the dispatching method and sends batch information to the load port under the scheduling module's instructions. In the following paragraphs, we introduce the FSM design of each component in a cluster tool.

1) Load Port: After receiving batches of wafers from the dispatching module, the load port checks whether the wafers are present in the load port or not. The wafers are then sent into the chambers sequentially. The load port continues sending and receiving wafers until all the wafers from this batch are processed and sent back to the load port.

Fig. 3 shows the FSM of the load port. The blue dotted lines indicate the signals received from the scheduling module. Each component's module needs instructions from the scheduling module when it exchanges the wafer information with the other modules. For instance, after receiving a wafer loading/unloading signal, the wafer information will be received/delivered to the robot arm. The load port also receives batch loading/unloading signals from the scheduling module, it then exchanges the batch information with the dispatching module.

Idle state represents that a load port is empty. After receiving a batch of wafers and conducting self checking, the load port will be at *Ready* state and start sending out wafers to be processed. At *Load*/*Unload* state, the wafer information will be exchanged with the robot arm. When there is no wafer that needs to be sent out, the load port will be at *Wait* state to collect the remaining wafers instead of going back to *Ready* state. After the whole batch is processed completely, the load port will be at *All done* state until the scheduling module notifies the load port to unload the batch.

2) Aligner: The tasks of an aligner are wafer loading, unloading, and aligning. The FSM of the aligner, as shown in Fig. 4(a), includes the loading/unloading procedure. After finishing aligning at *Aligning* state, the aligner will wait for the robot arm to pick up the wafer.



(a) Aligner

(b) Robot Arm

Fig. 4: The Finite State Machines for Aligner and Robot Arm.

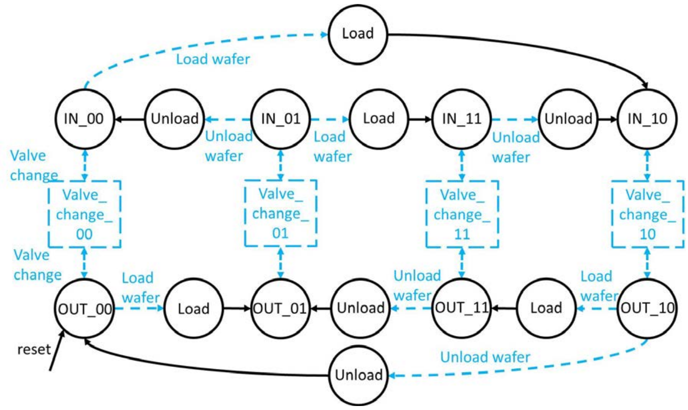


Fig. 5: The Finite State Machine for Load Lock.

3) Robot Arm: The robot arm is used to move wafers among the load ports, the aligner, and the load locks. As shown in Fig. 4(b), the FSM of the robot arm includes *Load* and *Unload* states. The robot arm receives and delivers the wafer information to its neighboring components: load port, aligner, and load lock. At *Transferring* state, the robot arm transfers the wafer to the destination. The robot arm holds the wafer at *Wait* state until the unloading signal is asserted from the scheduling module.

4) Load Lock: A load lock has two slots. One of them is for wafers returning back to the load ports, the other is for wafers heading towards the chambers. Our work uses one bit to represent the occupancy of slots. “1” represents that the slot is occupied and “0” represents that the slot is vacant. In Fig. 5, the last two bits of a state, from the left to the right, represent the slots for wafers returning back to the load ports and heading towards the chambers, respectively. The *IN* and *OUT* states refer to the gates of the load lock being opened on the chamber side and the load port side, respectively. For instance, *IN_10* represents that the load lock only has a wafer that will return back to the load port while the gate on the chamber side is opened.

The dotted-line boxes, *Valve_change* followed by two bits representing slot occupancy in Fig. 5, details the air pressure changes at the load lock. Using *Valve_change_00*, shown in Fig. 6, as an example, after the venting and pumping procedures, the load lock will stay at *Wait_OUT_00* and *Wait_IN_00* states. Since the decision on the gate opening is controlled by the scheduling module, the load lock will only move to the *Opening* state after receiving the signal from the scheduling module.

5) Buffer Arm: Before picking up a wafer, the buffer arm needs to move to the correct location. As shown in Fig. 7, after receiving a signal from the scheduling module, the buffer arm enters *Moving* state for its waferless rotation. The buffer arm is then ready to pick up a wafer at *Ready* state. After picking up a wafer and transferring it at *Transferring* state, the buffer arm stays at *Wait* state for the scheduling module’s unloading instruction.

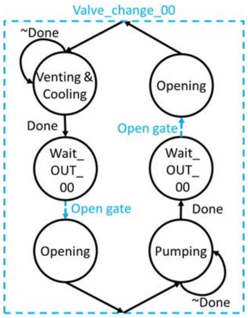


Fig. 6: The Finite State Machine for Valve_change_00.

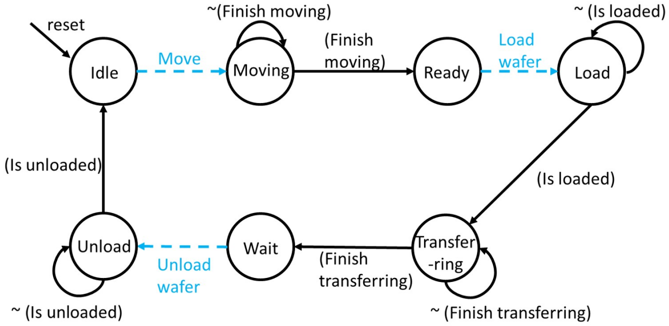


Fig. 7: The Finite State Machine for Buffer Arm.

6) Chamber: The chambers of the cluster tool have two tasks, wafer processing and cleaning. As shown in Fig. 8, the chamber starts at *Idle* state where it is waferless and its gate is closed. For processing wafer, the chamber opens the gate under the scheduling module’s instruction. After opening the gate, the chamber is ready to receive wafers at *Opened* state. The chamber starts processing wafer at *Processing* state after the wafer is loaded and the gate is closed. After processing, the chamber opens the gate and waits for the buffer arm to pick up the wafer at *Wait* state.

There are two time points for chamber cleaning. The first one is when the chamber stays at the *Idle* state for a period of time exceeding the time limit. The other is when the chamber receives the cleaning instruction from the scheduling module. The cleaning procedure is represented by *Clean* state in Fig. 8.

B. Scheduling

In this section, we discuss the tasks of the scheduling module, which includes the instructions for the robot arm and buffer arm movements, for gate opening at the chambers and the load locks, and for the resource allocation about a batch of wafers.

We propose two scheduling methods regarding the robot arm and buffer arm movements: *priority scheduling* and *fixed sequence scheduling*. For the *priority scheduling*,

TABLE I: An example for Priority Scheduling.

Priority	Load Lock	Buffer Arm	Chamber A	Chamber B	Instruction	Wafer Source	Wafer Destination
1	IN_01 / IN_11 / Wait_IN_01 / Wait_IN_11	Idle	Idle	-	Signal: Move To: Buffer Arm	Load Lock	Chamber A
2	IN_01 / IN_11 / Wait_IN_01 / Wait_IN_11	Idle	-	Idle	Signal: Move To: Buffer Arm	Load Lock	Chamber B
3	IN_01 / IN_00 / Wait_IN_01 / Wait_IN_00	Idle	Wait	-	Signal: Move To: Buffer Arm	Chamber A	Load Lock
4	IN_01 / IN_00 / Wait_IN_01 / Wait_IN_00	Idle	-	Wait	Signal: Move To: Buffer Arm	Chamber B	Load Lock

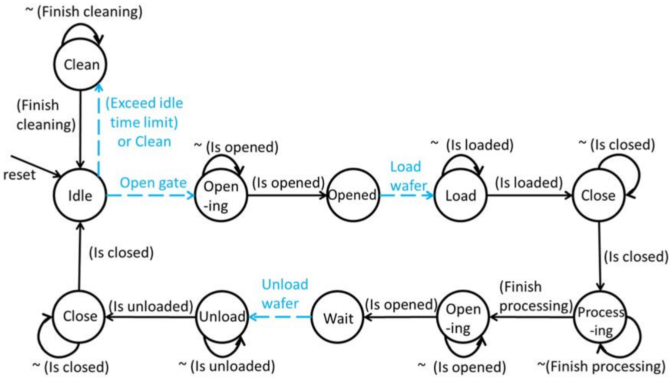


Fig. 8: The Finite State Machine for Chamber.

the scheduling module determines the movements of the robot arm and buffer arm by checking the states of their neighboring components. The scheduling module checks for three conditions before initiating an arm movement. The first one is to ensure that the arm is available, in other words, at *Idle* state. The second is that the wafer source component, where the wafer is currently located, is ready to unload the wafer. Finally, to prevent deadlocks, the scheduling module needs to ensure that the wafer destination component, where the wafer will be transferred, is ready to receive the wafer.

Assume that the cluster tool has one load lock and two chambers, Chamber A and Chamber B, and that the wafer in the load lock can be processed at any one of these chambers. With this configuration, TABLE I shows an example that lists the possible buffer arm movements satisfying the conditions mentioned above. Column 1 shows the priority number of the buffer arm movement. The *priority scheduling* arranges all the possible movements in an order, and assigns each movement a priority number, where the smaller number represents the higher priority. Columns 2 to 4 present the states of the components. The "-" in TABLE I means that the state of the component is irrelevant for this buffer arm movement. The *Instruction* column shows the signal sent by the scheduling module and the component where the signal is sent to. The *Wafer Source* column and the *Wafer Destination* column show where the wafer is currently located and the place the wafer will be transferred to, respectively. For instance, the first row shows the initiation of a buffer arm movement transferring the wafer from the load lock to Chamber A. The first condition is that the buffer arm needs to be at *Idle* state. The second condition is that the load lock has the wafer ready to be unloaded. The load lock module in the first row can be in *IN_01*, *IN_11*, *Wait_IN_01*, or *Wait_IN_11* states. All of these states represent that the load lock has a wafer heading towards the chambers for processing. Finally, Chamber A has to be available to receive the wafer. In other words, it has to be at *Idle* state. When all the conditions are satisfied, the scheduling module then sends *Move* instruction to the buffer arm. Note that the states shown in TABLE I can be changed under different cluster tools. For instance, in the first row, if the load lock's gate has to be opened for this buffer arm movement, *Wait_IN_01* and *Wait_IN_11* states will be excluded since both states represent that the load lock is with closed gates.

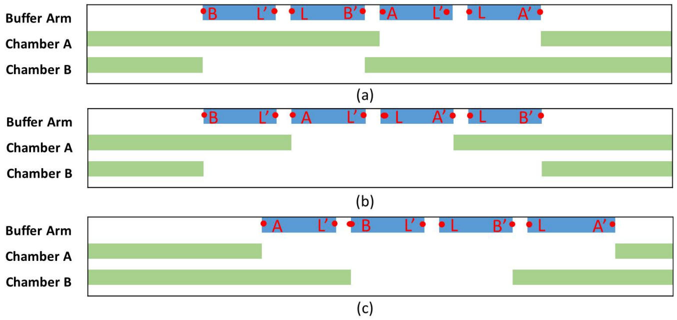


Fig. 9: Buffer arm movements with different *component strings*, where "L" represents the load lock, "A" represents Chamber A, and "B" represents Chamber B. (a) *Component string*: B-L'-L-B'-A-L'-L-A'. (b) *Component string*: B-L'-A-L'-L-A'-L-B'. (c) *Component string*: A-L'-B-L'-L-B'-L-A'.

For the *fixed sequence scheduling*, the scheduling module uses a *component string* to represent the sequence of components visited by an arm. The buffer arm and the robot arm have their corresponding *component strings*. Each component has a pair of characters in the string representing a wafer loading and unloading procedure at the component. Specifically, " A " indicates the wafer unloading procedure at Chamber A, and " A' " indicates the loading procedure at Chamber A.

Fig. 9. shows Gantt charts with different buffer arm movements. The rectangles represent the occupancy of wafers at the component. The row of Buffer Arm shows the sequence about chambers and load lock visited by the buffer arm. The sequence is represented by the *component string*. For instance, in Fig. 9(a), a wafer at Chamber B is unloaded and transferred

TABLE II: An example for Gate Opening in Load Lock and Chamber.

Load Lock	Buffer Arm	Buffer Arm Destination	Chamber	Instructions
Wait_IN_01 / Wait_IN_11	Moving	Load Lock	-	Load Lock: Open gate
Wait_IN_00 / Wait_IN_10	Transfer	Load Lock	-	Load Lock: Open gate
-	Transfer	Chamber	Idle	Chamber: Open gate

to the load lock (L) by the buffer arm. Then another wafer at the load lock is loaded and transferred to Chamber B by the same buffer arm. Next, the wafer at Chamber A is unloaded and transferred to the load lock, and another new wafer is transferred and loaded into Chamber A for processing. The corresponding *component string* is B-L'-L-B'-A-L'-L-A. The scheduling module refers to the *component strings* and the three conditions mentioned above when initiating an arm movement. For instance, the first row of TABLE I represents that the load lock has a wafer to be unloaded and transferred to Chamber A for processing. This scenario can be described with the *component string* containing L-A'. The scheduling module guides the buffer arm to move only when all the three conditions are met. With *fixed sequence scheduling*, a *component string* generated by any scheduling method can be tested on the simulator under the same cluster tool configuration.

The scheduling module guides the gates to be opened before a wafer is unloaded from chambers or load locks, or loaded into chambers or load locks. As shown in TABLE II, the scheduling module checks the state and destination of the buffer arm to determine if the gate should be opened at the chamber or load lock.

When a new batch of wafers arrives at the load port, the scheduling module needs to check if the resources such as chambers and load locks are available. If the chambers and load locks are available, the load port then starts sending out wafers to be processed. After a batch of wafers leaves the load port, the scheduling module checks if the current resources can be released for the batches at other load ports currently. The scheduling module keeps track of the relations between the batch of wafers and the resources. Before initiating any arm movement, the relations between the batch of wafers and the resources are checked. In other words, only the resources related to the batch are checked for the arm movement. For instance, for the buffer arm movements, the scheduling module only checks the states of chambers and load locks related to the batch.

C. Batch Dispatching

The dispatching module is responsible for batch dispatching. As shown in Fig. 10, at *Idle* state, the dispatching module checks if any batch is ready to be dispatched into the load port. The dispatching module enters *Ready* state with the chosen batch, then the batch is sent into the load port after receiving *Enable dispatch* from the scheduling module.

This framework accepts two dispatching methods, *self-dispatch* and *dispatch on demand*. For the *self-dispatch*, the dispatching module uses the input dispatching time point given by engineers to dispatch the batch. When the scheduling module confirms that the dispatching module is at *Ready* state, and the assigned load port is available for the batch, it sends *Enable dispatch* signal to the load port for loading the batch.

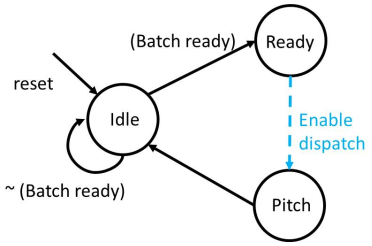


Fig. 10: The Finite State Machine for the dispatching module.

TABLE III: Batch information of Cluster Tool X.

Batch Name	Number of wafers	Process Chamber
X Batch 1	25	Chamber A / Chamber B
X Batch 2	4	Chamber C / Chamber D
X Batch 3	25	Chamber C / Chamber D
X Batch 4	25	Chamber A / Chamber B
X Batch 5	25	Chamber A / Chamber B
X Batch 6	25	Chamber C / Chamber D
X Batch 7	25	Chamber A / Chamber B
X Batch 8	25	Chamber D

For the proposed *dispatch on demand*, the dispatching module receives the request for a new batch of wafers from the scheduling module when the resources are available. For this method, our framework gives users two options regarding the dispatching order. One is to dispatch based on the batch ordering. The other is to dispatch the batch that can acquire the most available resources. With *dispatch on demand*, the dispatching time point of each batch is gathered during the simulation. After the simulation, the dispatching time point is recommended to engineers. Since a new batch is requested immediately after the resources become available with this method, it greatly reduces the idle time of those resources. With less idle time at the chambers, the cluster tool processes these batches of wafers more efficiently. Therefore, a higher throughput of a cluster tool is achieved.

IV. Experimental Results

The proposed cluster tool simulation framework was implemented in C++ language with SystemC. The log file information was provided by a semiconductor manufacturer in Taiwan. The Gantt charts were generated by using the plotly Python library [24].

The experiments were performed on a cluster tool, Cluster Tool X. The cluster tool is parallel mode single-armed with three load ports, one aligner, one robot arm, two load locks, one buffer arm, and four chambers. The batch information used in the simulation is shown in TABLE III. The *Process Chamber* column shows the chambers where the batches can be processed. Most of the batches can be processed at either chambers A and B, or chambers C and D. Note that X Batch 2 is a test batch containing only four wafers.

The comparisons of experimental results about Cluster Tool X and its simulations are summarized in TABLE IV. The *Scheduling Method* column and *Dispatching Method* column show the methods used in the simulation. *Dispatch on demand* can only be paired with *priority scheduling* since the arm movement sequence cannot be predicted beforehand. The *Processing Time* column represents the time duration between the first wafer being unloaded from the load port and the last wafer being loaded into the load port. With *T(Simulation)* denoting to the processing time of the simulation and *T(Real)* denoting to the processing time of the cluster tool, the *Error Rate* column is calculated using the following equation:

$$Error\ Rate\ (\%) = \frac{|T(Simulation) - T(Real)|}{T(Real)}$$

The simulation results of *priority scheduling* and *fixed sequence scheduling* have 1.91% and 1.95% error rates, respectively. The processing time of simulation with *dispatch on demand* is eight hours and seven seconds, 43 minutes and 11 seconds shorter than the real processing time. In other words, the simulation using *dispatch on demand* yields a higher throughput than the cluster tool. The simulator then returns the dispatching time points to engineers for improving the throughput of the cluster tool in the succeeding runs. The recommended dispatching time points of Cluster Tool X batches are shown in TABLE V. Using the first batch dispatching time point as a reference point, the dispatching time points for the other batches are shown using |Batch dispatching time - First batch dispatching time|.

The Gantt charts of data generated by Cluster Tool X and the simulation result with *dispatch on demand* are shown in Fig. 11 and Fig. 12, respectively. A, B, C, and D represent the chambers. Load_Lock1_s1 and Load_Lock2_s1 represent the slots for wafers heading towards the chambers, and Load_Lock1_s2 and Load_Lock2_s2 represent the slots for wafers returning to the load ports. The Gantt charts show that when a batch of wafers can be processed at two chambers, the wafers are placed alternately between these two chambers. By dispatching a new batch immediately after the resources become available, *dispatch on demand* reduces the idle time between batches. Since the idle time between batches are reduced, the batches of wafers in Fig. 12 are more compressed in time with *dispatch on demand* as compared to that for Cluster Tool X.

Note that the simulation time of all the experiments is less than one second, while the actual processing time of cluster tools is more than eight and a half hours. Hence, our simulator provides a relatively accurate result with high efficiency.

V. Conclusion

In this paper, we discuss the functionality of each component in a cluster tool, and propose a flexible simulation framework on which a variety of cluster tools can be implemented. With *priority scheduling*, the scheduling decisions can be described using the states of components. *Priority scheduling* allows the scheduling logic to be observed by users. The *fixed sequence scheduling* method provides the option for different schedulings to be deployed on the same cluster tool. The proposed dispatching method, *dispatch on demand*, provides the dispatching time points of batches for achieving a higher throughput. Finally, the experimental results show that the proposed simulation framework can estimate the whole processing time with a small error rate in terms of time difference and can perform simulation with different wafer paths easily.

References

[1] Tung-He Chou, Yu-Chih Chang, Binglung Yu, Chun-Fu Chen, Yung-Tai Hung, Tuung Luoh, Ling-Wuu Yang, Tahone Yang, and Kuang-Chao Chen, "Capacity Simulation by Cellular Automation in Endura Platform," in *Proc. e-Manufacturing and Design Collaboration Symposium*, 2016.

[2] Henry L. Gantt, "A graphical daily balance in manufacture," *Transactions of the American Society of Mechanical Engineers*, vol 24, pp. 1322–1336, 1903.

[3] Taehee Jeong, Deeksha Prakash Kankalale, Raymond Chau, and Hyeran Jeon, "Going Deeper or Wider: Throughput Prediction for Cluster Tools with Machine Learning," in *Proc. International Conference on Big Data Computing, Applications and Technologies*, 2019.

[4] Chihyun Jung and Tae-Eog Lee, "An Efficient Mixed Integer Programming Model Based on Timed Petri Nets for Diverse Complex Cluster Tool Scheduling Problems," *IEEE Transactions on Semiconductor Manufacturing*, 2012, vol. 25, no. 2, pp. 186-199.

[5] Chihyun Jung and Tae-Eog Lee, "Cyclic Scheduling of Cluster Tools with Nonidentical Chamber Access Times Between Parallel Chambers," *IEEE Transactions on Semiconductor Manufacturing*, 2012, vol. 25, no. 3, pp. 420-431.

[6] Dae-Kyu Kim, Tae-Eog Lee, and Hyun-Jung Kim, "Optimal Scheduling of Transient Cycles for Single-Armed Cluster Tools," in *Proc. International Conference on Automation Science and Engineering*, 2013, pp. 874-879.

[7] Ja-Hee Kim, Tae-Eog Lee, Hwan-Yong Lee, and Doo-Byeong Park, "Scheduling Analysis of Time-Constrained Dual-Armed Cluster Tools," *IEEE Transactions on Semiconductor Manufacturing*, 2003, vol. 16, no. 3, pp. 521-535.

[8] Ja-Hee Kim and Tae-Eog Lee, "Schedulability Analysis of Time-Constrained Cluster Tools with Bounded Time Variation by an Extended Petri Net," *IEEE Transactions on Automation Science and Engineering*, 2008, vol. 5, no. 3, pp. 490-504.

[9] Sung-Gil Ko, Tae-Sun Yu, and Tae-Eog Lee, "Scheduling Dual-Armed Cluster Tools for Concurrent Processing of Multiple Wafer Types Witw Identical Job Flows," *IEEE Transactions on Automation Science and Engineering*, 2019, vol. 16, no. 3, pp. 1058-1070.

[10] Jun-Ho Lee, Hyun-Jung Kim, and Tae-Eog Lee, "Scheduling Cluster Tools for Concurrent Processing of Two Wafer Types," *IEEE Transactions on Automation Science and Engineering*, 2014, vol. 11, no. 2, pp. 525-536.

[11] George H. Mealy, "A method for synthesizing sequential circuits," *Bell Labs Technical Journal*, vol 34, pp.1045-1079, 1955.

[12] Edward F. Moore, "Gedanken-Experiments on Sequential Machines," *Automata studies*, pp. 129–153, 1956.

[13] David A. Nehme and Neal G. Pierce, "Evaluating the throughput of cluster tools using event-graph simulations," in *Proc. Advanced Semiconductor Manufacturing Conference and Workshop (ASMC)*, 1994, pp. 189–192.

[14] Kyungsu Park and James R. Morrison, "Controlled Wafer Release in Clustered Photolithography Tools Flexible Flow Line Job Release Scheduling and an LMOLP Heuristic," *IEEE Transactions on Automation Science and Engineering*, 2015, vol. 12, no. 2, pp. 642-656.

[15] Sang C. Park, Euikoog Ahn, Yongho Chung, Ka-ram Yang, Byung H. Kim, and Jeong C. Seo, "Fab Simulation with Recipe Arrangement if Tools," in *Proc. Winter Simulations Conference (WSC)*, 2013, pp. 3840-3849.

[16] Paht Te Quek, Boon Ping Gan , Song Lian Tan, Chan Lai Peng, and Bart vd Heijden, "Analysis of the front-end wet strip efficiency performance for productivity," in *Proc. International Symposium on Semiconductor Manufacturing*, 2007, pp. 1–4.

[17] Shadi Rostami, Babak Hamidzadeh, and Dan Camporese, "An Optimal Periodic Scheduler for Dual-arm Robots in Cluster Tools with Residency Constraints," *IEEE Transactions on Robotics and Automation*, 2001, vol. 17, no. 5, pp. 609-619.

TABLE IV: Processing time of Cluster Tool X and simulation results.

Approach	Scheduling Method	Dispatching Method	Processing Time	Error Rate (%)
Cluster Tool X	-	-	8 hours 43 minutes 18 seconds	-
Simulation	*Priority Scheduling*	*Self-dispatch*	8 hours 33 minutes 4 seconds	1.91
	Fixed Sequence Scheduling		8 hours 33 minutes 16 seconds	1.95
	Priority Scheduling	*Dispatch on Demand*	8 hours 0 minutes 7 seconds	-

TABLE V: Recommended dispatching time points of Cluster Tool X.

Batch Name	|Batch dispatching time - First batch dispatching time|
X Batch 1	0 hours 0 minutes 0 seconds
X Batch 2	0 hours 0 minutes 0 seconds
X Batch 3	0 hours 13 minutes 49 seconds
X Batch 4	1 hours 13 minutes 27 seconds
X Batch 5	2 hours 14 minutes 38 seconds
X Batch 6	2 hours 30 minutes 15 seconds
X Batch 7	3 hours 46 minutes 57 seconds
X Batch 8	4 hours 12 minutes 15 seconds

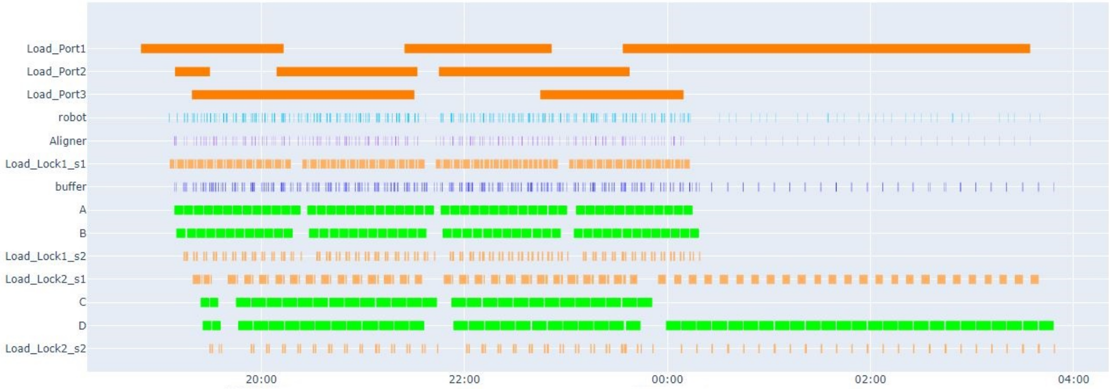


Fig. 11: The Gantt chart of Cluster Tool X.

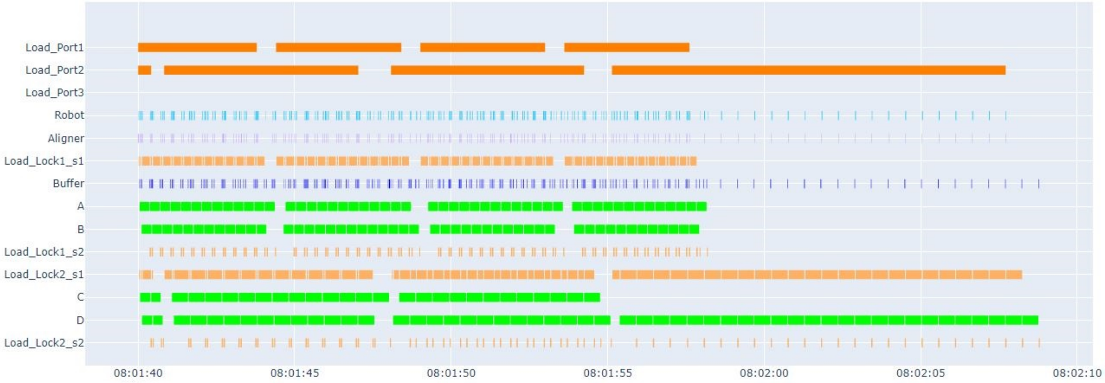


Fig. 12: The Gantt chart of simulation with Cluster Tool X batch information using *Dispatch on Demand*.

[18] Hiroe Watanabe, “Development of Wafer Transfer Simulator Based on Cellular Automata,” *IEEE Transactions on Semiconductor Manufacturing*, 2015, vol. 28, no. 3, pp. 283-288.

[19] Uno Wikborg and Tae-Eog Lee, “A Petri Net Method for Schedulability and Scheduling Problems in Single-Arm Cluster Tools with Wafer Residency Time Constraints,” *IEEE Transactions on Semiconductor Manufacturing*, 2008, vol. 21, no. 2, pp. 224-237.

[20] Uno Wikborg and Tae-Eog Lee, “Noncyclic Scheduling for Timed Discrete-Event Systems with Application to Single-Armed Cluster Tools Using Pareto-Optimal Optimization,” in *Proc. International Conference on Automation Science and Engineering*, 2013, vol. 10, no. 3, pp. 699-710.

[21] Kan Wu, Ning Zhao, and Carman K. M. Lee, “Queue Time Approximations for a Cluster Tool with Job Cascading,” *IEEE Transactions on Automation Science and Engineering*, 2016, vol. 13, no. 2, pp. 1200-1207.

[22] Xiuhong Zheng, Jingtao Hu, and Haibin Yu, “Research on a simulation platform assisting analysis of the cluster tool,” in *Proc. International Conference on Computer Research and Development*, pp. 249-253, 2011.

[23] Wlodek M. Zuberek, “Cluster Tools with Chamber Revisiting—Modeling and Analysis Using Timed Petri Nets,” *IEEE Transactions on Semiconductor Manufacturing*, 2004, vol. 17, no. 3, pp. 333-344.

[24] Plotly Technologies Inc., “Collaborative data science,” Available: https://plot.ly.