(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 May 2009 (07.05.2009)

PCT

(10) International Publication Number
WO 2009/058002 A1

(51) International Patent Classification:
G05B 19/045 (2006.01)

(21) International Application Number:
PCT/NL2008/000243

(22) International Filing Date: 30 October 2008 (30.10.2008)

(25) Filing Language: Dutch

(26) Publication Language: English

(30) Priority Data:
1034599 30 October 2007 (30.10.2007) NL

(71) Applicant *(for all designated States except US)*: **UNIVERSITEIT TWENTE** [NL/NL]; Drienerloolaan 5, NL-7522 NB Enschede (NL).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **GÜLESIR, Gürcan** [TR/NL]; Jupiterstraat 103, NL-7521 JK Enschede (NL). **BERGMANS, Louis, Marie, Johannes** [NL/NL]; De Ijsvogel 10, NL-7491 ZG Delden (NL). **AKSIT, Mehmet** [NL/NL]; Reygershöftehoek 99, NL-7546 KD Enschede (NL).

(74) Agents: **LAND, Addick, Adrianus, Gosling** et al.; Arnold & Siedsma, Sweelinckplein 1, NL-2517 GK The Hague (NL).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD FOR AN EVENT-DRIVEN SYSTEM

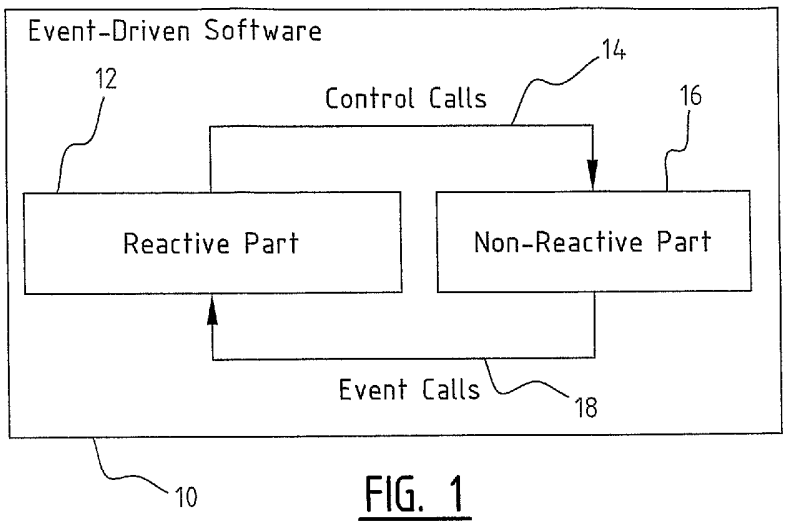

FIG. 1

(57) Abstract: Method for an event-driven system with an initial state and at least one state transition from a source state to a destination state and an input alphabet with at least one input symbol, wherein an input symbol from the input alphabet is associated with a state transition, which state transition represents the transition from a source state to a destination state when the input symbol associated with the state transition is received, wherein at least one state transition is associated with all input symbols not already associated with one of the other state transitions sharing the same source state.

WO 2009/058002 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

Method for an event-driven system

The present invention relates to a method for an event-driven system. The present invention also relates to software for specifying software for an event-driven system. The present invention further relates to a system for specifying software for an event-driven system. In addition, the present invention relates to software specified as according to such a method. The present invention also relates to a data carrier comprising software according to the invention.

Photolithographic techniques are used in the semiconductor industry for the production of semiconductor products. During the photolithographic process patterns which must be arranged on or in the semiconductor material, wafer 21, are imaged on a light-sensitive layer on the semiconductor surface in a lithographic projection apparatus 20, a so-called wafer scanner. For this purpose light 27 from a laser source 26 is shone through a mask 23 on which the pattern for imaging is arranged, after which the light 27 incident through mask 23 is cast onto the light-sensitive layer on the semiconductor surface using a lens 28. Because simultaneous display of the fine structures of the pattern on the whole semiconductor surface is not possible, the semiconductor surface is divided into segments 29 which are illuminated successively. Nor is the whole of a single segment 29 illuminated simultaneously here, but a scanning movement is performed over the segment surface in order to eventually illuminate the whole segment surface. For this purpose mask 23 and the semiconductor surface are moved through light beam 27 in mutually opposite directions. Figure 2 also shows mask table 24, wafer table 22 and the pattern 25 to be imaged on mask 23.

Among other factors, the combination of the fine structures to be imaged and the high speed at which wafer scanner 20 operates in order to achieve the highest possible productivity result in a complex control of wafer scanner 20. A wafer scanner 20 typically comprises about 400 sensors

and 300 actuators, and the software for controlling wafer scanner 20 comprises about fifteen million lines of source code. In order to keep the writing and maintaining of the source code manageable, use is made of various methods for structuring the source code and the writing and maintaining of the source code.

A frequently used method for structuring such extensive software systems is to sub-divide the software into a reactive part and a non-reactive part. The reactive part models different possible states of the system. This part receives events from, among other parts, the sensors and the non-reactive part. On the basis of these events the reactive part determines which control calls must be sent to the non-reactive part and into which new state the system enters. The non-reactive part is responsible for performing services which are called by the reactive part.

In most cases the reactive part can be modelled using a state chart. The state chart comprises a number of states corresponding with possible states of the system for controlling. State transitions always connect a source state to a destination state and have an associated input symbol which represents an event. When an event is received the state transition, which has the present state as source state and which is associated with the input symbol which represents the received event, determines the (destination) state to which the system moves. The subject of state charts itself will not be further elaborated here since the skilled person is familiar with state charts.

The complexity and size of software for the purpose of for instance controlling lithographic projection equipment entails modifications being made during the lifespan of the software, for instance in respect of software maintenance or product improvement, wherein errors may be made which result in defects in the software and the system.

Examples of such defects are:

- an event call is defined but there is no corresponding event;

- there is an event but the corresponding event call is not defined;
- the reactive part expects an event call which never takes place;
- the target (function, method, procedure) of a control call is absent;
- the non-reactive part does not provide a required service.

It is self-evident that, as the software becomes larger and more complex, manual correction of such defects becomes an impossible task.

The present invention has for its object to provide a method with which determining of the above described defects is simplified, and such defects can even be prevented.

This object is achieved with the invention by providing a method for an event-driven system with an initial state and at least one state transition from a source state to a destination state and an input alphabet with at least one input symbol, wherein an input symbol from the input alphabet is associated with a state transition, which state transition represents the transition from a source state to a destination state when the input symbol associated with the state transition is received, wherein at least one state transition is associated with all input symbols not already associated with one of the other state transitions sharing the same source state. The specifying of the event-driven system on the basis of a state chart makes it possible to formulate a precise specification of the system. In traditional state charts only a single input symbol is associated with a state transition. In the present invention an extra symbol is introduced, the context-sensitive wildcard, which can be associated with a state transition and which represents one or more input symbols, in contrast to the traditional method of a single input symbol being associated with a state transition. Which symbol is represented depends on the other state transitions which share the same source state as the state transition

with the context-sensitive wildcard. The context-sensitive wildcard represents (for this state transition) all input symbols which are not already associated with one of these other state transitions, including input symbols which at the moment of specification did not (yet) form part of the input alphabet. It hereby becomes possible to define state transitions for input symbols which are introduced only a later time into the life cycle of the software, for instance as a result of the addition of new functionalities. A greater expressivity is hereby obtained than in traditional state charts.

The present invention further provides a method wherein at least one output symbol is associated with a state and wherein the method further comprises of: generating an output which is represented by the output symbol associated with the state of the event-driven system. In this way the specification of the output corresponds with a Moore machine.

The invention also provides a method wherein at least one output symbol is associated with a state transition, and the method further comprises of: generating an output which is represented by the output symbol associated with the state transition from the source state to the destination state when the event-driven system moves from a source state to a destination state. The output is now defined in a manner comparable to a Mealy machine.

The present invention provides a further method wherein the event-driven system comprises at least one sensor, and wherein at least one input symbol represents a representation of a detection by the sensor. It is possible in this manner for the system to react to measurements on objects for processing and to input provided by operators.

The present invention also provides a method wherein the event-driven system comprises at least one actuator and wherein the output symbol represents a control signal for the actuator. This extension enables the system to for instance manipulate objects for processing.

In a further embodiment of the invention an event-driven system is provided, wherein the output symbol represents a reproduction by means of a reproducing device. The reproduction can be perceived by an operator who obtains information about the event-driven system on the basis of the reproduction information. Examples of such a reproduction are an alert that the system has completed an operation or a confirmation of an operating command.

Yet a further embodiment provides an event-driven system wherein the output symbol comprises the graphic representation of a symbol on a reproducing device. In a specific embodiment the reproducing device comprises a screen. This screen can for instance show the progress of the process executed by the event-driven system.

The present invention further provides a method wherein the event-driven system is a lithographic projection apparatus. Using sensors the position and orientation of the wafer is for instance detected. Stepping motors are then for instance used to correctly position and orient the wafer. Another sensor makes it possible to establish whether contaminants are present on the mask, while an actuator is arranged to clean the mask if it is determined that the mask is not clean.

According to the invention a method is also provided wherein at least one input symbol represents a function call. This makes it possible to specify the sequence of activities. In another method a function call must here be understood in a broad sense. Finally, the source code of the function can alternatively by incorporated directly in the calling function, whereby one cannot speak of a function call. In this case function call must be understood to mean a block of statements which execute a clearly delimited task, such as for instance cleaning of the mask or positioning of the wafer.

The invention further provides a method, further comprising of: specifying an event by specifying a condition under which the event occurs, comprising of defining at least one state and a state transition from a source state

to a destination state with an input symbol from the input alphabet associated with the state transition; and of linking an event call to a state which represents a state of the event-driven system in which the event call takes place. The condition for the event comprises for instance a required series of operations which must be executed in a fixed sequence. In another example the condition comprises of a single detection by a sensor. In yet another example the condition comprises of a combination of detections by sensors and operations which must occur or take place in a determined sequence. This method has the additional advantage that the specification specifies the states in which an event call must be generated, whereby it becomes possible to have the implementation of the event call in the source code take place automatically. As will be shown later in this description with reference to the figures, a specific part of the source code of software can be associated with each state. This association also provides the option of determining the location in the source code where the event call must be generated. On the basis of the specification it is thus possible, without any contribution from a developer, to determine the location of the event call, and to actually insert this event call. The chance of an implementation which differs from the specification is hereby reduced, which decreases the risk of errors in the software.

The invention further provides a method wherein at least one state is defined as the final state, the method further comprising of: verifying whether a source code meets the given specification, comprising of: determining all possible flows of control of the source code; running through the state charts for each flow of control of the source code in order to determine whether the flow of control results in a state defined as a final state; and determining whether a state defined as final state is reached for all possible flows of control. Source code is here understood to mean one or more series of related statements, irrespective of whether they implement a full

application, a single module, a single library or a single function or routine.

According to this method, not only is a system specified but whether the given source code meets the specification is also verified. Possible variations from the specification can then be submitted to a developer in order to determine the problem: is the specification incorrect or is the implementation incorrect. The developer can then correct the specification or the source code, after which the source code is once again checked on the basis of the specification. This method has the advantage that the chance of detecting errors in the software is considerably increased.

The present invention also provides a method wherein the possible flows of control of the source code are modelled by a simplified control flow graph (SCFG). The advantage of the SCFG modelling the possible flows of control is that this model does not depend on the programming language used to implement the system. It is hereby possible to make the algorithm for verifying the source code independently of the implementation language, whereby the algorithm can be used for more than a single implementation language.

The present invention further provides a method wherein the simplified control flow graph is constructed by first constructing an abstract syntax tree (AST) of the source code. In contrast to the SCFG, the AST is not implementation-independent but assists in the construction of the SCFG by providing a systematic, hierarchical division of the source code, on the basis of which the construction of the SCFG is facilitated.

The present invention also provides a method wherein the verification further comprises of constructing a mapping of a state with which an event is associated at a position in the source code for verifying; and of inserting an event call of the associated event at the mapped position. Use is preferably made in this step of the SCFG and the AST for finding the relation between the states from

the specifications and the associated positions in the source code. On the basis of this relation the event call of an event is inserted in the source code, whereby the event calls establishes a link between the non-reactive part of the software and the reactive part. In a preferred embodiment of the invention this step is executed automatically in order to obviate the chance of an error by the developer.

In a preferred embodiment the present invention provides software for specifying software which, if run on a processor, executes a method as described above.

In another embodiment the present invention provides a system for specifying software according to a method as described.

In an alternative embodiment the present invention provides software specified according to a method as described.

In yet another embodiment the invention provides a device comprising software specified according to a method as described.

The present invention further provides a data carrier comprising such software.

Further advantages and embodiments of the present invention will become apparent on the basis of the following description with reference to the figures, in which:

Figure 1 shows a schematic representation of software to which a method according to the present invention can be applied;

Figure 2 shows a schematic view of a wafer scanner, to the control software of which a method according to the present invention can be applied;

Figure 3 shows a simplified control flow graph of the wafer scanner to which a method of the present invention can be applied;

Figure 4 shows a schematic representation of a method according to the present invention;

Figure 5 shows a specification in accordance with a method according to the present invention;

Figure 6 shows a specification in accordance with a method according to the present invention;

Figure 7 shows another specification in accordance with a method according to the present invention;

Figure 8 shows yet another specification in accordance with a method according to the present invention;

Figure 9 shows an abstract syntax tree and a simplified control flow graph of a simplified function of control software of the wafer scanner to which a method according to the present invention can be applied; and

Figure 10 shows another abstract syntax tree of a simplified function of control software of the wafer scanner to which a method according to the present invention can be applied.

Figure 11 shows yet another abstract syntax tree of a simplified function of control software of the wafer scanner to which a method according to the present invention can be applied.

In the control of wafer scanner 20 (figure 2) the successive positioning and scanning of each wafer segment 29 are the actual processing steps (processing) of wafer 21. Before these processing steps can actually begin, a number of preprocessing steps must be executed. Due to the great precision required in the production of the semiconductor products, relatively small irregularities and disruption can quickly result in defective products. In order to reduce the chance of defective products the mask 23 must be as clean as possible and the shape imperfections of wafer 21 must be known as well as possible. For this purpose preprocessing steps are executed, such as cleaning the mask and measuring the shape imperfections of wafer 21.

In order to produce the highest possible number of operating semiconductor products, the preprocessing steps are steps which must necessarily be carried out before the actual processing steps may be executed. This relation can be expressed in a state chart (figure 3).

The wafer scanner begins initially 31 by transposing 32 to a state of readiness READY 33. In the

state of readiness READY 33 the wafer scanner waits until a start event is received, which event can for instance be initiated by operating personnel pressing a button. Receiving the start event causes the wafer scanner to move 34 from the state of readiness READY 33 to the preprocessing state PREPROCESSING 35. As soon as the preprocessing is completed, a preprocessed event (preprocessed) ensures that transition 36 to the processing state PROCESSING 37 is initiated. As soon as the processing steps are also completed a processed event (processed) is generated, whereby wafer scanner moves 38 to the final state 39 and stops.

The most extensive method according to the invention basically comprises four steps (figure 4).

Step I: The software developer derives and specifies the compatibility constraints on the basis of, among other things, the software requirements, the state charts and the implementations of the various tasks of the software.

Step II: The developer specifies the events and links the event calls.

Step III: A source code analyser analyses the source code on the basis of the specifications from steps I and II and produces inter alia a specification of the determined compatibility errors. The developer repairs the defect on the basis of this specification of a compatibility error. The source code analyser also produces a list of event points with corresponding event calls.

Step IV: On the basis of the source code and the list of event points with corresponding event calls produced by the source code analyser a source code to source code converter inserts the corresponding event call at each event point in the source code.

Once steps I and II have been executed, steps III and IV can be executed repeatedly after modification of the source code so as to guarantee that the software does not have the above stated defects. The four steps are discussed in more detail hereinbelow.

Step I, the specification of the conditions, will be discussed in the following paragraphs.

Figure 5 shows an example of a specification of a compatibility constraint. The shown exemplary compatibility constraint C1 requires that the wafer be measured during the preprocessing of the wafer. The specification specifies the order of acceptable sequences of function calls. A large rectangle 50 represents the source code, designated with label 52 in this example of the preprocessing function *preprocess*. The label «from» is a specific example of a designation for the range of the applicability of the specification. In this case the label indicates that the specification applies to all calls *from* the preprocessing function *preprocess*. The preprocessing function has two states q0, q1. A first state q0 is the initial state of the preprocessing function, this being designated with label «initial» 53. The state transition *measureWafer* 54 specifies a call to the *measureWafer* function. If the preprocessing function *preprocess* 52 calls the function *measureWafer*, state q1 is then reached. The specification further shows two state transitions 56, 57 which are provided with a dollar sign ($). The dollar sign is the context-sensitive wildcard which indicates that all symbols (function calls) are associated with this state transition, with the exception of the symbols already associated with a state transition which shares the same source state as the state transition with which the context-sensitive wildcard is associated. In this embodiment all function calls, except for the call to *measureWafer*, are thus associated with state transition 56 because *measureWafer* is already associated with state transition 54. All function calls are without exception further associated with state transition 57.

State q1 is further provided with label «final» 55, which indicates that this state is a valid state for the function *preprocess* to end in. All function call sequences from the function *preprocess* which do not end in state q1 do not therefore meet the specification of figure 5 because only state q1 is a valid final state, i.e. provided with the

label «final». The valid function call sequences in the function *preprocess* are all sequences in which the function *measureWafer* is called at least once. Possible other function calls in the sequence correspond with either state transition 56, if they precede the (first) function call *measureWafer* 54, or state transition 57 if they take place after the (first) function call.

A second compatibility constraint C2 for the preprocessing function is that the mask is optionally cleaned, for instance if a sensor, such as for instance a camera, has determined that mask 23 comprises contaminants, but that this cleaning may not take place after measuring of wafer 21. In the sequence of function calls this translates into the preprocessing function *preprocess* not being allowed (any longer) to call the cleaning mask function *cleanReticle* after the measure wafer function *measureWafer* has been called.

Note that this compatibility constraint C2 does not require wafer 21 having to be measured without limitation (irrespective of whether the mask is cleaned). This requirement is after all already set in compatibility constraint C1. The specification of compatibility constraint C2 described hereinbelow will therefore not include this constraint, since compatibility constraint C1 is specified by its own specification. The skilled person will however appreciate that it is also unconditionally possible to construct a single specification in which both compatibility constraints C1 and C2 are specified.

Compatibility constraint C2 is specified as follows (figure 6): The specification once again specifies function call sequences «from» (label 51) the preprocessing function *preprocess* (label 52). The initial state is (label 63) state q0, this state also being a valid final state. As long as the measure wafer function *measureWafer* has not yet been called from the preprocessing function *preprocess*, any random function can be called (including the cleaning mask function *cleanReticle*), as indicated by state transition 65, with which the context-sensitive wildcard is associated. As

indicated above, this specification does not require the presence of a call to the measure wafer function *measureWafer* in the function call sequence, since state q0 is a valid final state. If the preprocessing function *preprocess* does however call the measure wafer function *measureWafer*, state q1 is then reached via state transition 66. If the cleaning mask function *cleanReticle* is called after the measure wafer function *measureWafer* has been called, state q2 is then reached via state transition 68.

This state is not a valid final state, nor can a valid final state be reached from this state. Calling the cleaning mask function *cleanReticle* after calling the measure wafer function *measureWafer* thus produces a function call sequence which does not meet this specification. It is however permissible to call a function other than the cleaning mask function *cleanReticle* after the measure wafer function *measureWafer* has been called, as specified by state transition 57 with which the context-sensitive wildcard is associated.

In the example of the preprocessing function *preprocess* the preprocessed event *preprocessed* is generated by calling the event call *preprocessed*() once the preprocessing function has been completed. How this event is specified in step II and how the event call is linked will be described hereinbelow.

The preprocessed event *preprocessed* is only generated if the final function call in preprocessing function *preprocess* is the call to the function *measureWafer* (figure 7). An initial start is made in the initial state q0. The label «initial-final» 73 specifies that state q0 is the initial state, but also a valid final state.

Note that states are specific to a single specification. States with the same designation in different specifications (and here different figures) are not necessarily the same identical states.

All function call sequences are valid sequences in this example, since all conditions, i.e. q0 and q1, are all final states. The intention of the specification in figure 7

is not to specify the valid function call sequences but only to specify when and in which state the preprocessed event call *preprocessed*() is called. This is specified by coupling this event to state q1 by means of specifying the function call *preprocessed*() 79 as output symbol in state q1. Figure 7 now specifies that, each time state q1 is reached, preprocessed event *preprocessed* is generated by calling *preprocessed*(). This is the case if the function *measureWafer* is called via state transitions 74 or 78. Any other random function call corresponds with the context-sensitive wildcard ($) and results, via one of the two state transitions 76 and 77, in state q0 in which no event is generated.

The above described compatibility constraints C1 and C2 and the specification of preprocessed event *preprocessed* can also be specified in a single specification (figure 8). Function call sequences are once again shown «from» (label 51) the preprocessing function *preprocess* (label 52). The initial state is q0, as specified by label 83. State transition 84 specifies that all functions other than the measure wafer function *measureWafer* (but including cleaning mask function *cleanReticle*) may be called from q0, wherein there is a return in each case to state q0. State q0 is however not a valid final state (label 83) and in order to reach a valid final state measure wafer function *measureWafer* must be called (state transition 85). In state q1 the preprocessed event *preprocessed* is then generated by event call *preprocessed*(); label 86. Through further calls to the measure wafer function *measureWafer* and calls to functions other than the cleaning mask function *cleanReticle* and the measure wafer function *measureWafer* the respective states q1 and q2 are reached, both of which are valid final states and therefore produce function call sequences which are in accordance with the specification. State q3 is however reached by a call to the cleaning mask function *cleanReticle* after the measure wafer function *measureWafer* has already been called. This state is not a valid final state and no valid final states can be reached from this

state. A call to the cleaning mask function *cleanReticle* after the measure wafer function *measureWafer* has been called thus produces a call sequence which does not meet the specification. According to the specification the function call sequence must at least comprise a call to the measure wafer function *measureWafer* and no further calls to the cleaning mask function *cleanReticle* may take place after this function call. After each call to the measure wafer function *measureWafer* a preprocessed event *preprocessed* is further generated by calling the preprocessing function *preprocessed*();.

In step III the source code is analysed on the basis of the specifications of steps I and II. For this example use is made of the following source code for the preprocessing function *preprocess* in programming language C:

```
1    void preprocess()
2    {
3         if (!reticleClean)
4         {
5              cleanReticle();
6         }
7         measureWafer();
8    }
```

The source code analyser analyses the preprocessing function *preprocess* and constructs an abstract syntax tree (AST) (upper part of figure 9). The node *FDef* represents a function definition. The nodes *FCall* represent function calls.

If the compatibility constraints and the events can be specified on the basis of function calls and the possible flow of control, only a part of the data from the AST is then necessary and on the basis of the AST a simpler model can be constructed which models only the function calls and the flow of control between the calls, i.e. a simplified control flow graph (SCFG) (lower part of figure 9). An additional advantage of the SCFG compared to the AST is that the SCFG is not dependent on the chosen implementation language.

The SCFG is derived by traversing the AST according to the depth-first search. The broken arrows indicate the corresponding points between the AST and the SCFG. The black dot to the left in the SCFG is the initial node, which represents the start of the implementation of the preprocessing function *preprocess*. The cleaning mask node *cleanReticle* 95 represents the call to the cleaning mask function *cleanReticle*. The measure wafer node *measureWafer* 97 represents the function call to the measure wafer function *measureWafer*. Finally, the black dot to the right represents the end of the implementation of preprocessing function *preprocess*. This dot is circled in order to indicate that it is an end node. The SCFG thus specifies the possible function call sequences which are possible on the basis of the source code.

Once the SCFG has been constructed, the source code analyser generates a list of possible function call sequences from the function for analysing, in this case preprocessing function *preprocess*. For this purpose the SCFG is traversed in accordance with the depth-first search. It can be readily appreciated in this case that there are only two possible sequences, i.e. the call to only the measure wafer function *measureWafer* and the call to the cleaning mask function *cleanReticle* followed by the call to the measure wafer function *measureWafer*. By running through the specifications on the basis of these two function call sequences (figure 8) the source code analyser determines that a valid final state is reached for all function call sequences, i.e. in this case state q1. The source code analyser concludes herefrom that the source code meets the compatibility constraints C1 and C2. If the source code analyser has found function call sequences which did not comply with one of the compatibility constraints (this being concluded because a valid final state is not reached in the specification), an error message is then generated having therein the function call sequence(s) which does not comply with the constraints.

For the purpose of the following step (step IV) a mapping of the states in the specification (figure 8) to the nodes in the SCFG is constructed during the comparison of the function call sequences possible on the basis of the SCFG. In the embodiment discussed here, state q0 from the specification is reproduced at node *cleanReticle* 95 and state q1 is (according to both function call sequences) reproduced at node *measureWafer* 97.

In step IV a source code to source code transformation is executed in order to incorporate into the source code the event calls specified in step III. The event specifications specify (step II) for all event calls in which state these calls must take place. The location in the source code where the relevant event call must be inserted can be determined on the basis of the mapping of states to nodes in the SCFG determined in step III.

According to the specification of figure 8, the event call for the preprocessed event *preprocessed*() 86 is associated with state q1. In step III is determined that state q1 corresponds with node *measureWafer* 97 in the SCFG (lower part of figure 9). Node *measureWafer* 97 in the SCFG corresponds with the function call *measureWafer()* in the AST (upper part of figure 9). The event call for the preprocessed event *preprocessed*() (figure 10) must thus be incorporated in the AST (figure 11) after the function call *measureWafer()*. This results in the event call for the preprocessed event *preprocessed*() also having to be incorporated in the source code after the function call *measureWafer()*. Step IV hereby produces the following source code:

```
1    void  preprocess()
2    {
3         if (!reticleClean)
4         {
5              cleanReticle() ;
6         }
7         measureWafer();
8         preprocessed();
```

```
9    }
```

With this final step the process is completed. A specification is formulated of the conditions which the preprocessing function *preprocess* must satisfy. An event has been specified. The source code is then verified on the basis of these specifications, and an event call for the specified event is finally added to the source code.

The embodiments according to the invention described and shown in the description and figures are only exemplary embodiments. The skilled person will appreciate that many changes and modifications are possible which fall within the present invention. It will thus be apparent to the skilled person that the sequences of activities do not necessarily have to be defined by function calls, but that any other indication of delimitation of activities can be used for this purpose, such as for instance a comment designated in accordance with a predetermined convention. The protection sought is therefore not limited by the exemplary embodiments given here, but is defined by the following claims.

Claims

1. Method for an event-driven system with an initial state and at least one state transition from a source state to a destination state and an input alphabet with at least one input symbol,

wherein an input symbol from the input alphabet is associated with a state transition, which state transition represents the transition from a source state to a destination state when the input symbol associated with the state transition is received,

wherein at least one state transition is associated with all input symbols not already associated with one of the other state transitions sharing the same source state.

2. Method as claimed in claim 1, wherein at least one output symbol is associated with a state and wherein the method further comprises of:

generating an output which is represented by the output symbol associated with the state of the event-driven system.

3. Method as claimed in claim 1, wherein at least one output symbol is associated with a state transition, and the method further comprises of:

generating an output which is represented by the output symbol associated with the state transition from the source state to the destination state when the event-driven system moves from a source state to a destination state.

4. Method as claimed in claim 1, 2 or 3, wherein the event-driven system comprises at least one sensor, and

wherein at least one input symbol represents a representation of a detection by the sensor.

5. Method as claimed in claim 2, 3 or 4, wherein the event-driven system comprises at least one actuator, and

wherein the output symbol represents a control signal for the actuator.

6. Method as claimed in any of the claims 1-5, wherein the event-driven system is a lithographic projection apparatus (20).

7. Method as claimed in any of the claims 1-6, wherein at least one input symbol represents a function call.

8. Method as claimed in any of the claims 1-7, further comprising of:

specifying an event by specifying a condition under which the event occurs, comprising of defining at least one state and a state transition from a source state to a destination state with an input symbol from the input alphabet associated with the state transition; and

linking an event call to a state which represents a state of the event-driven system in which the event call takes place.

9. Method as claimed in claim 8, wherein at least one state is defined as final state, the method further comprising of: verifying whether a source code meets the specification given as according to claim 8, comprising of:

determining all possible flows of control of the source code;

running through the state charts for each flow of control of the source code in order to determine whether the flow of control results in a state defined as a final state; and

determining whether a state defined as final state is reached for all possible flows of control.

10. Method as claimed in claim 9, wherein the possible flow of control of the source code is modelled by a simplified control flow graph (SCFG).

11. Method as claimed in claim 10, wherein the simplified control flow graph is constructed by first constructing an abstract syntax tree of the source code.

12. Method as claimed in claim 9, 10 or 11, wherein the verification further comprises of constructing a mapping of a state with which an event is associated at a position in the source code for verifying; and

inserting an event call of the associated event at the mapped position.

13. Software for specifying software which, if run on a processor, executes a method as claimed in any of the claims 1-12.

14. System for specifying software according to a method as claimed in any of the claims 1-12.

15. Software specified according to a method as claimed in any of the claims 1-12.

16. Device comprising software as claimed in claim 15.

17. Data carrier comprising software as claimed in claim 13.

18. Data carrier comprising software as claimed in claim 15.

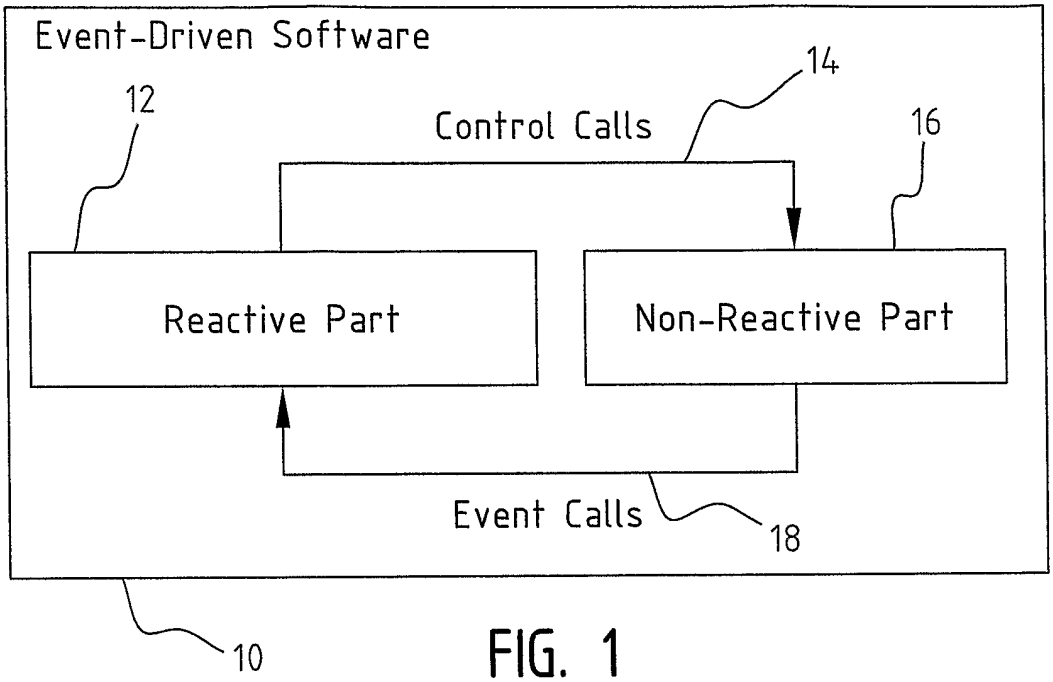
Event-Driven Software
12
14
Control Calls
16
Reactive Part
Non-Reactive Part
Event Calls
18
10

FIG. 1

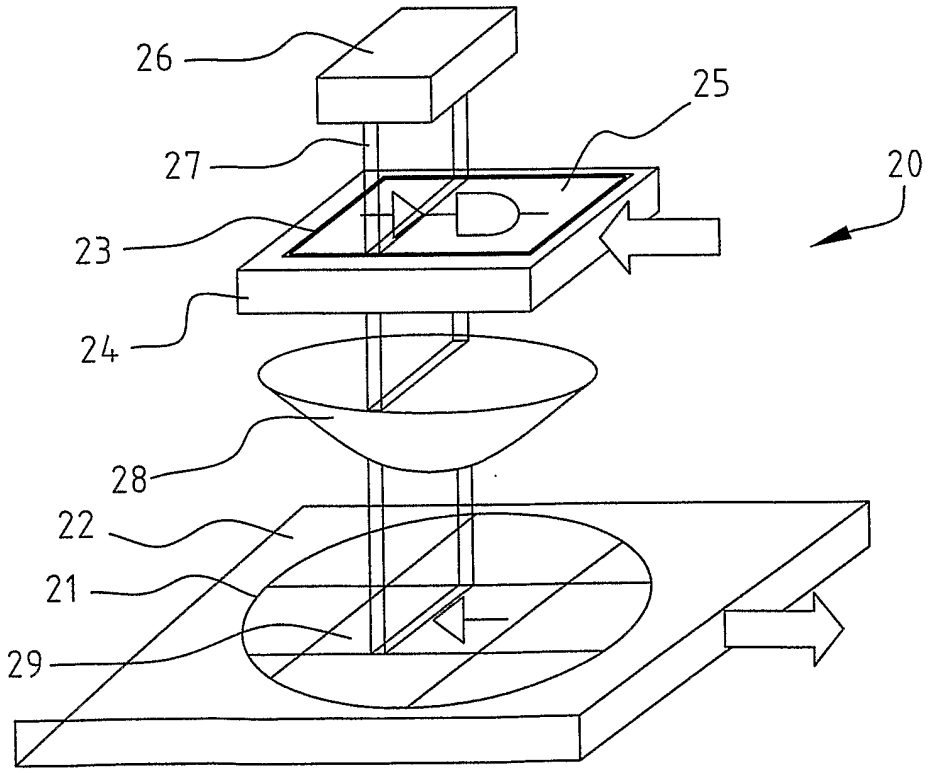
26
25
27
20
23
24
28
22
21
29

FIG. 2

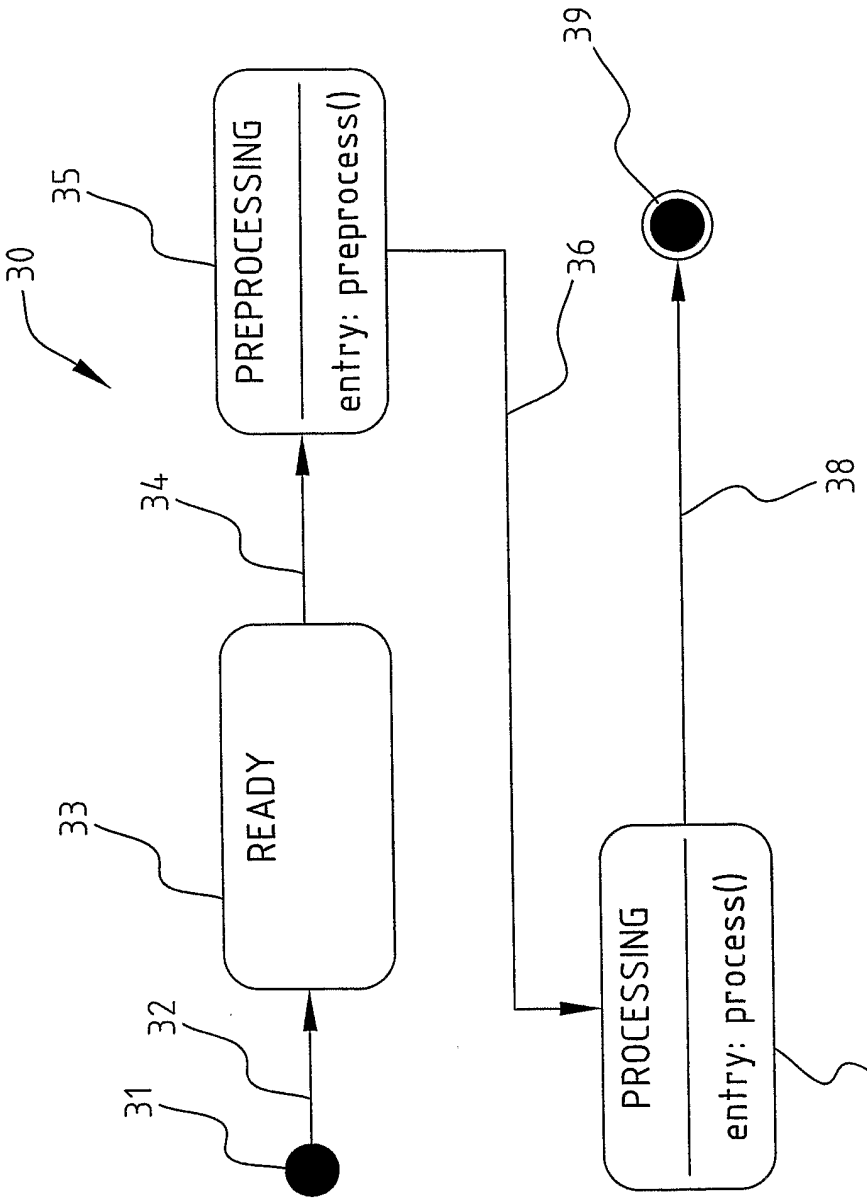
30
31
32
33
READY
34
35
PREPROCESSING
entry: preprocess()
36
37
PROCESSING
entry: process()
38
39

FIG. 3

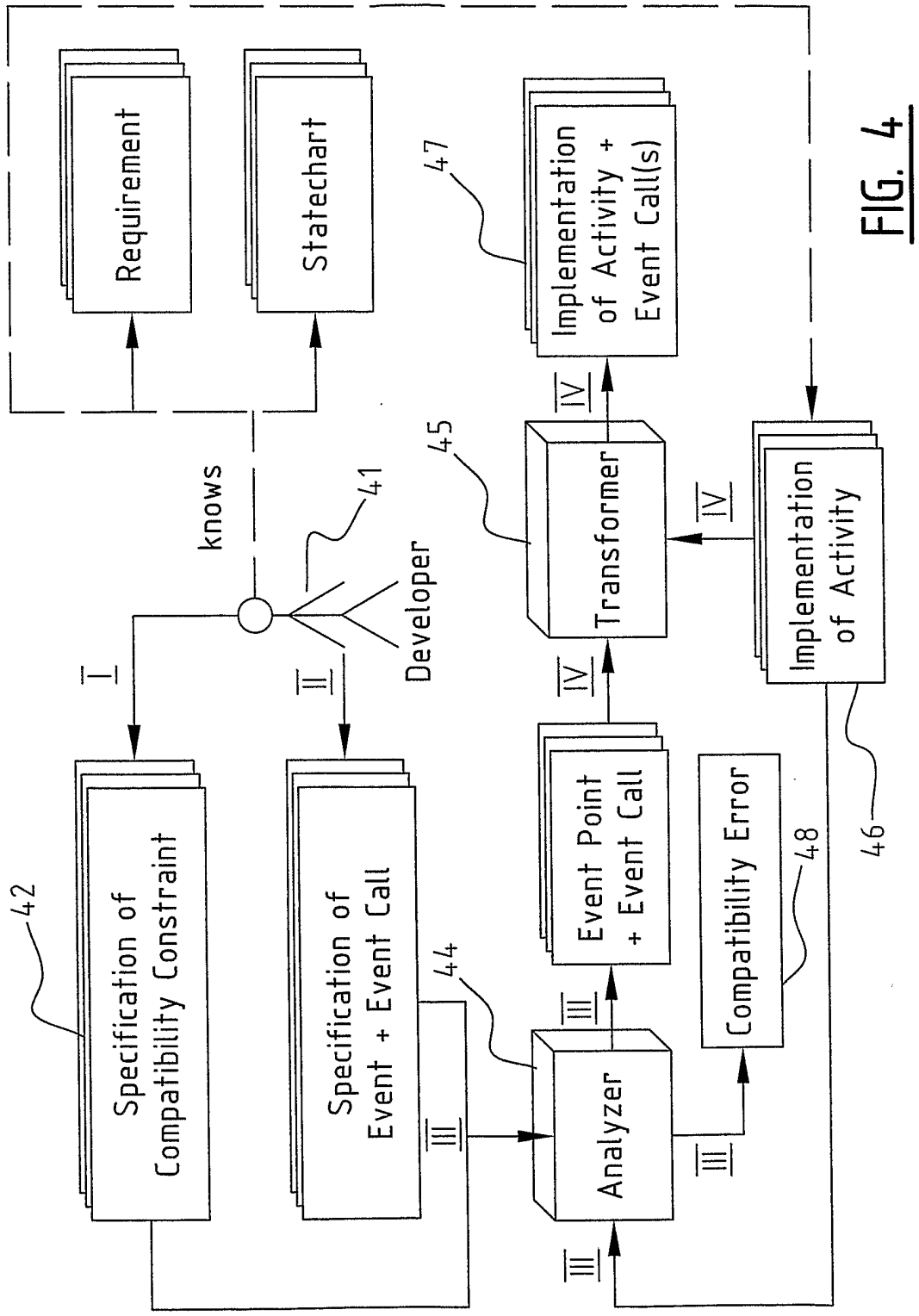
Requirement
Statechart
Implementation of Activity + Event Call(s)
47
knows
41
Developer
45
Transformer
IV
Implementation of Activity
I
II
IV
IV
42
Specification of Compatibility Constraint
Specification of Event + Event Call
44
Event Point + Event Call
Compatibility Error
48
46
III
III
III
III
Analyzer

FIG. 4

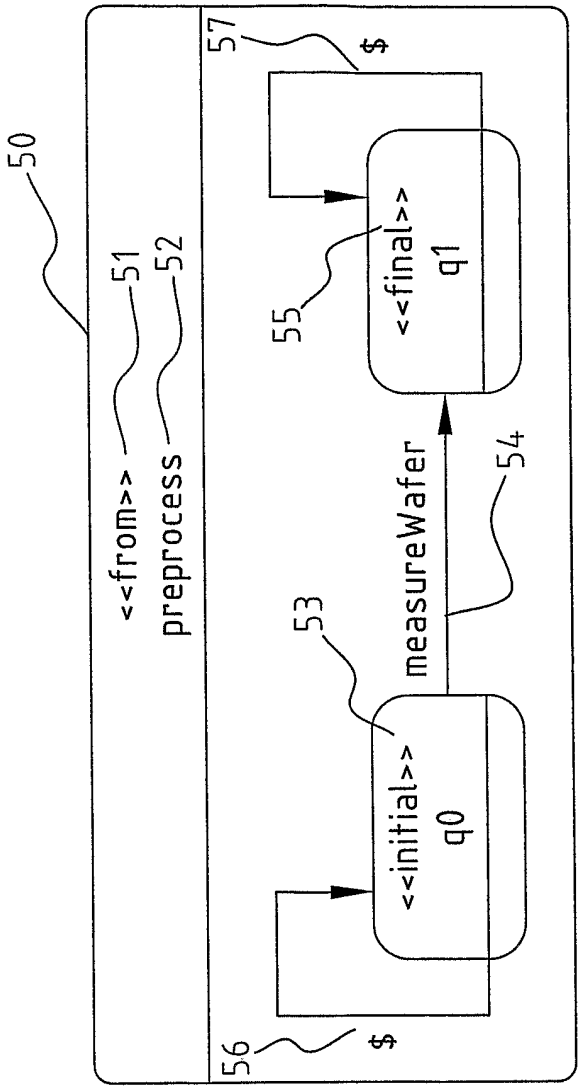
50
51
52
<<from>>
preprocess
57
$
55
<<final>>
q1
measureWafer
54
53
<<initial>>
q0
56
$

FIG. 5

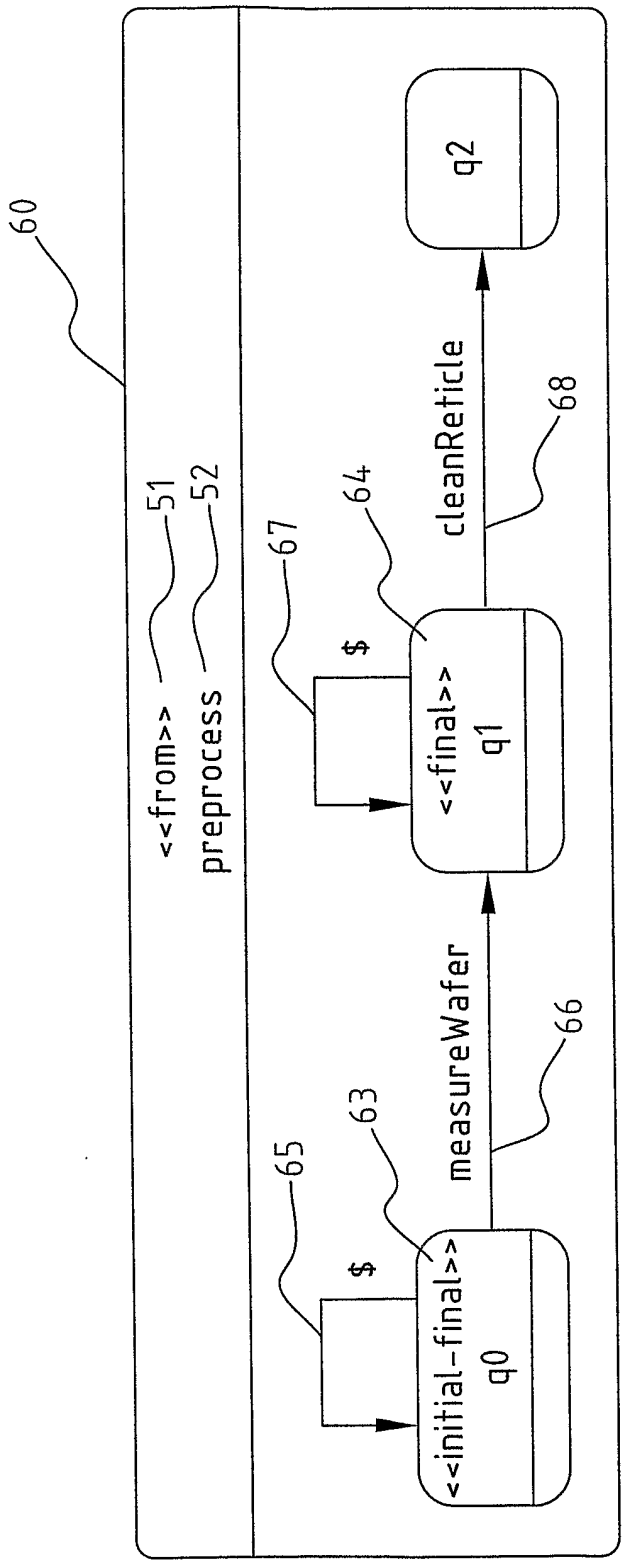
60
<<from>>
preprocess
51
52
67
$
64
<<final>>
q1
cleanReticle
68
q2
65
$
63
<<initial-final>>
q0
measureWafer
66

FIG. 6

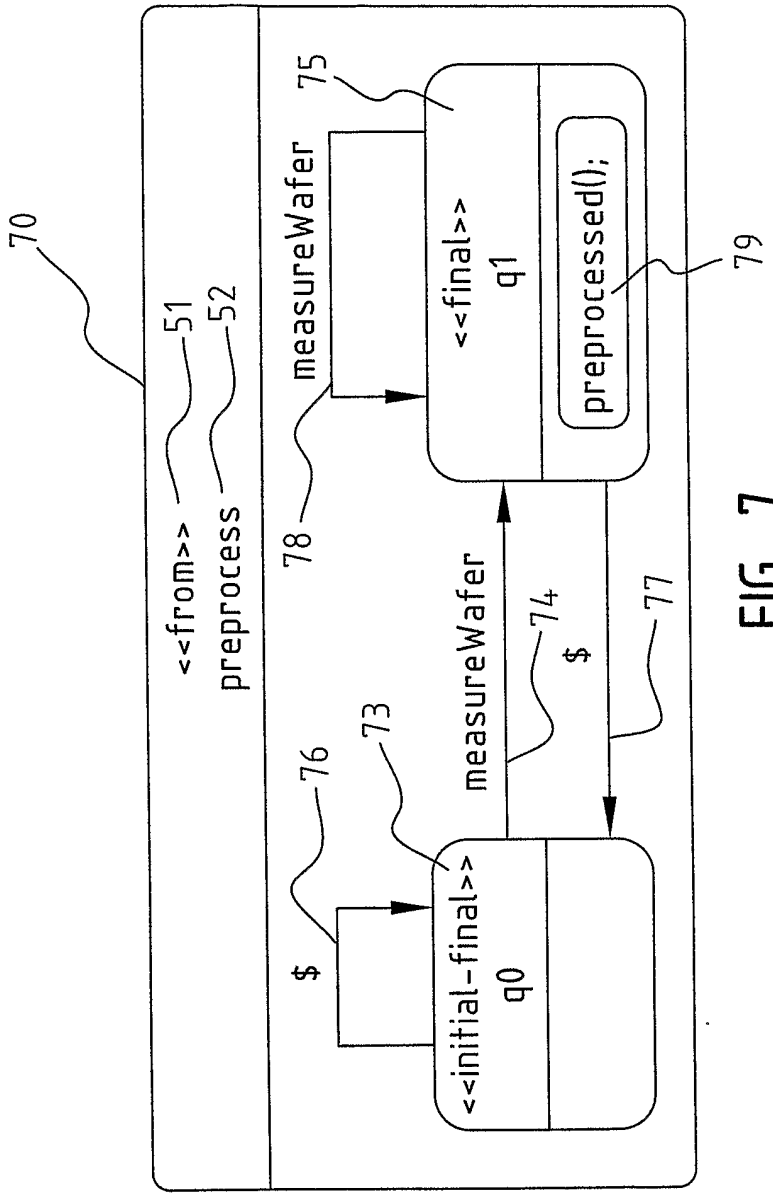
70
51
52
<<from>>
preprocess
75
measureWafer
78
<<final>>
q1
preprocessed();
79
measureWafer
74
$
77
76
$
73
<<initial-final>>
q0

FIG. 7

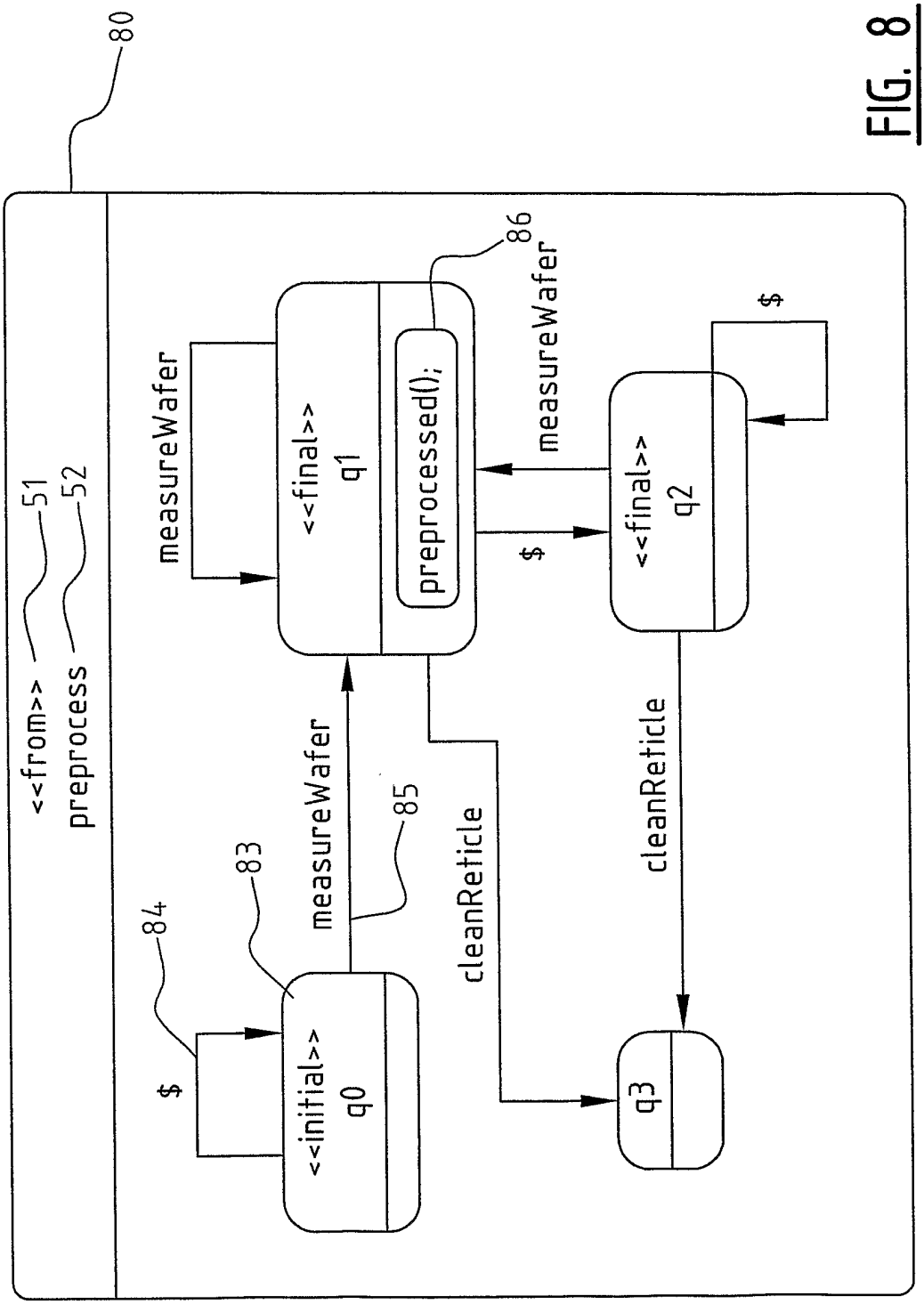
80
<<from>>
preprocess
51
52
measureWafer
<<final>>
q1
preprocessed();
86
measureWafer
$
<<final>>
q2
$
$
84
83
<<initial>>
q0
measureWafer
85
cleanReticle
cleanReticle
q3

FIG. 8

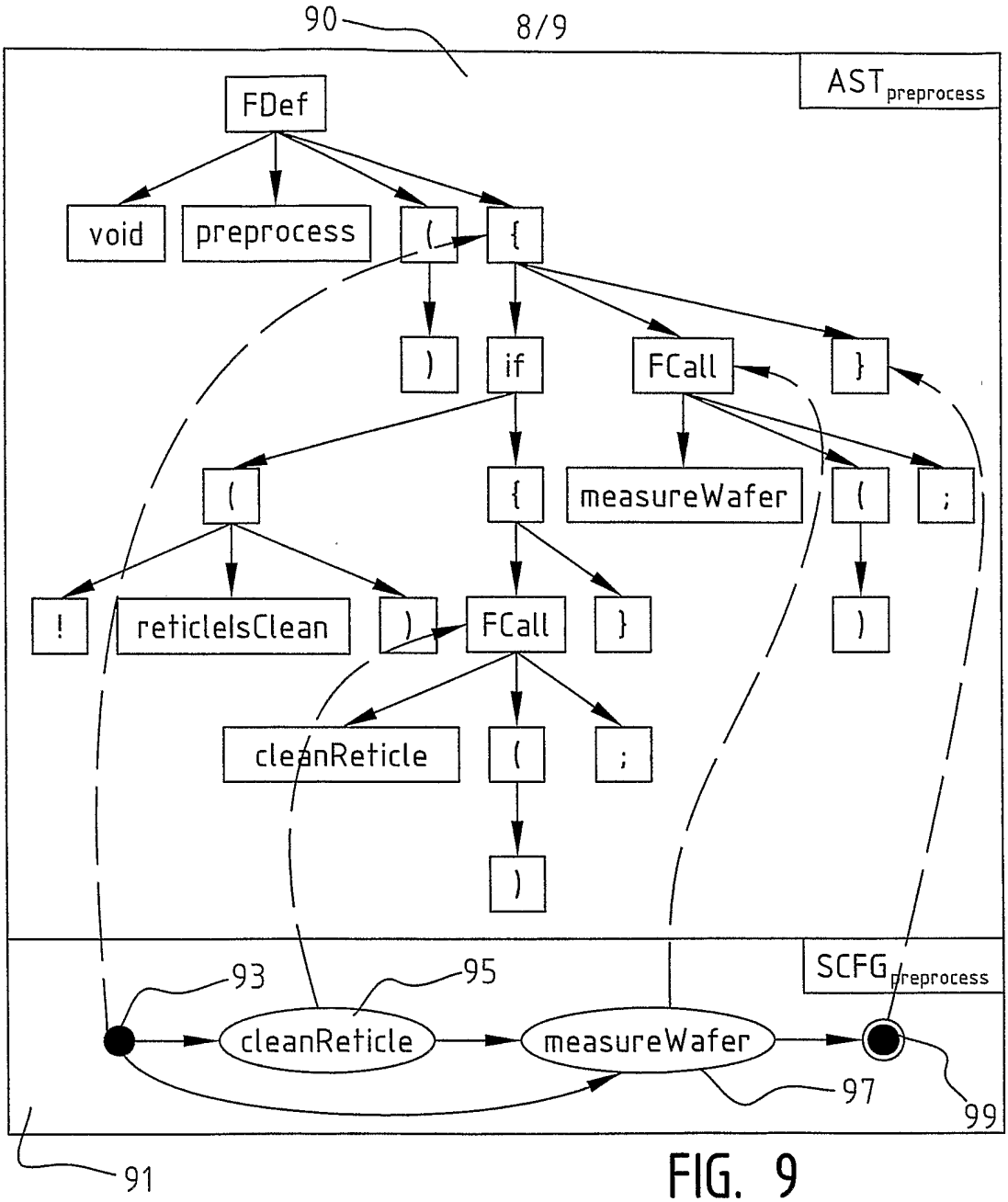
90
8/9
AST preprocess
FDef
void
preprocess
(
{
)
if
FCall
}
(
{
measureWafer
(
;
!
reticleIsClean
)
FCall
}
)
cleanReticle
(
;
)
SCFG preprocess
93
95
cleanReticle
measureWafer
97
99
91

FIG. 9

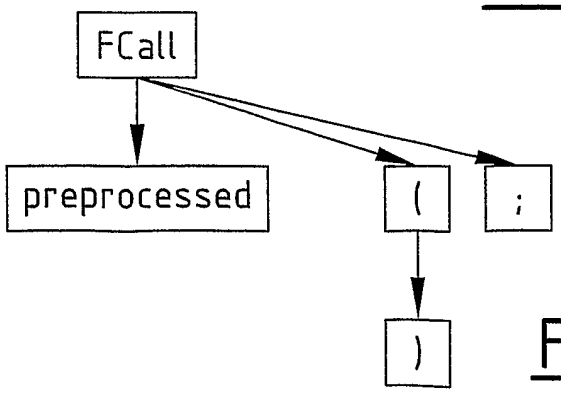
FCall
preprocessed
(
;
)

FIG. 10

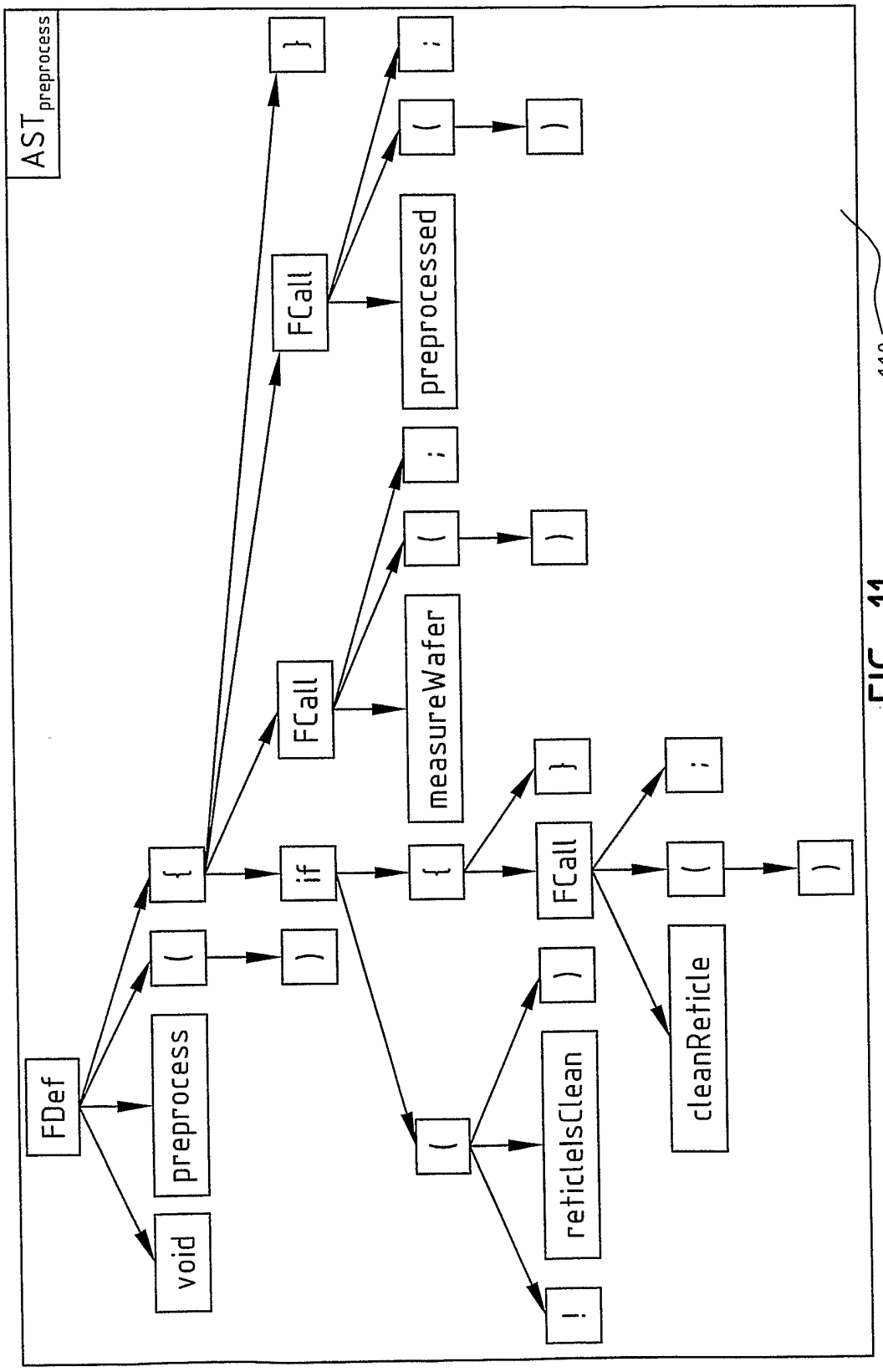
AST preprocess
FDef
void
preprocess
(
)
{
if
(
)
{
FCall
measureWafer
(
)
;
FCall
preprocessed
(
)
;
}
(
reticleIsClean
)
!
FCall
cleanReticle
(
)
;
}
110

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No
PCT/NL2008/000243

A. CLASSIFICATION OF SUBJECT MATTER
INV. G05B19/045

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G05B

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y A	US 6 170 024 B1 (WAKELAND CARL K [US] ET AL) 2 January 2001 (2001-01-02) figure 3 column 8, lines 48-51	1,8, 13-18 2-7 9-12
Y	US 2004/193290 A1 (OTT MICHAEL [US] ET AL) 30 September 2004 (2004-09-30) column 4, line 35 - column 8, line 52	2-7
X	US 5 905 902 A (O'CONNOR DENNIS [US]) 18 May 1999 (1999-05-18) paragraphs [0030] - [0032]	1
A	WO 2004/021181 A (VIHANA INC [US]; SHARANGPANI HARSHVARDHAN [US]) 11 March 2004 (2004-03-11) the whole document	1-18

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 March 2009	20/03/2009

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Fax: (+31–70) 340–3016	Groen, Fokke

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/NL2008/000243

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6170024 B1	02-01-2001	NONE	
US 2004193290 A1	30-09-2004	CN 1534420 A	06-10-2004
		DE 102004014747 A1	18-11-2004
		GB 2400190 A	06-10-2004
		HK 1069642 A1	11-05-2007
		JP 2004303247 A	28-10-2004
US 5905902 A	18-05-1999	NONE	
WO 2004021181 A	11-03-2004	AU 2003270044 A1	19-03-2004
		EP 1532496 A2	25-05-2005
		JP 2005537550 T	08-12-2005
		KR 20050083667 A	26-08-2005
		US 2004059443 A1	25-03-2004