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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 November 2016 (03.11.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/174132 A1

(51) International Patent Classification:
G06F 11/36 (2006.01) *G06F 17/50* (2006.01)

(21) International Application Number:
PCT/EP2016/059479

(22) International Filing Date:
28 April 2016 (28.04.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
15165729.3 29 April 2015 (29.04.2015) EP

(71) Applicant: **AREVA NP SAS** [FR/FR]; 1 place Jean Millier, 92400 Courbevoie (FR).

(72) Inventors: **KRECKEL, Richard**; Burgbergstr. 25, 91054 Erlangen (DE). **KÜNZEL, Andreas**; Bayernstr. 2, 91074 Herzogenaurach (DE).

(74) Agent: **KUGLER, Jörg**; Eschersheimer Landstr. 105-107, 60322 Frankfurt am Main (DE).

(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, BN, BR, BW, BY, BZ, CA, CH, CL, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, 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, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) Title: METHOD FOR CHECKING EQUIVALENCE OF CODE

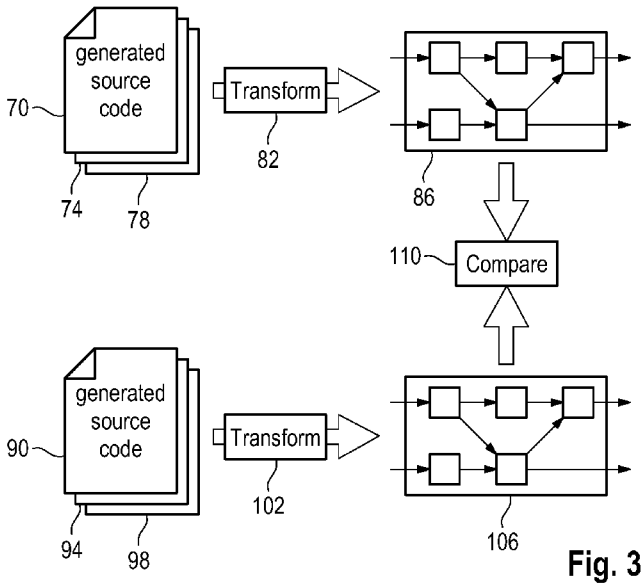


Fig. 3

(57) Abstract: A method for checking the functional equivalence of two pieces of software for control systems, in particular control systems based on programmable logic controllers, engineered from function block diagrams (120, 196) with a plurality of interconnected function blocks, the method comprising the following steps: a) generating a first instance of compilable source code from a first plurality of function block diagrams (70, 74, 78) by virtue of a first code generator obeying to a first set of sequentialization rules; b) parsing said first instance of source code and identifying the set of all function block I/O ports and junction points (nodes) and the set of all connections between them (arcs) of the data flow; c) using the information from step b) to reconstruct a first data flow graph (86) of said first instance of source code in the upstream direction; d) generating a second instance of compilable source code from a second plurality of function block diagrams (90, 94, 98) by virtue of the same or a second code generator obeying a second set of sequentialization rules; e) parsing said second instance of source code and identifying the set of all function block I/O ports and junction points (nodes) and the set of all connections between them (arcs) of the data flow; f) using the information from step e) to reconstruct a second data flow graph (106) of said second instance of source code in the upstream direction; g) comparing said first identified data flow graph (86) and said second identified data flow graph (106) with each other, node by node, and thereby checking if the transitive closure of input up to that node in said second instance of source code is the same as in said first instance of source code.

WO 2016/174132 A1

Description

Method for checking equivalence of code

Field of the Invention

The invention relates to a method for checking the functional equivalence of two pieces of software for safety control systems. It relates to control systems with source code automatically generated from function block diagrams. It also relates to a non-transitory computer-readable storage device. It also relates to a non-transitory computer-readable storage medium. It also relates to a computer program product, a system and a computer.

Background Art

For the safe and reliable operation of nuclear power plants or other safety-critical systems, electricity and control systems are of paramount importance, especially safety instrumentation & control (I&C) systems. For a given digital safety I&C system with possibly multiple redundant divisions and an engineering system which lets the system engineer specify the software using FBDs (function block diagrams) in multiple subsequent versions, the software needs to be validated for each redundant division and for each version until all acceptance criteria are met.

The overhead of validating each redundant division and each version is considerably large if one considers that there will be significant similarities: Some FBDs will be basically the same in different redundant divisions, only some FBDs will have changed from one version to the next one. Manual modifications to FBDs (on different redundant divisions or from one version to another) have a tendency to alter the generated code in a way that amplifies the differences in the generated code between these versions. This makes it generally impossible to pinpoint the differences in the generated codes.

Up to now all FBDs have to undergo verification and validation (V&V) activities even if their functionality was unchanged. During development or verification and

validation of application software, the task might arise to verify that two FBDs are functionally equivalent in order to omit redundant verification of validation activities for functionally equivalent FBDs. The need for this task might be caused by the need to identify function block diagrams which implement the same functions in different redundancies or the need to identify function block diagrams that have changed between different versions of an application.

In order to determine, whether one diagram implements the same functionality as another diagram, a simple line-by-line diff of the source codes of the two diagrams can raise many false positives. This is because the code generator has several degrees of freedom when generating source code for function block diagrams. There might be differences in the source codes ranging from different identifiers for symbols up to a different sequentialization of the function block calls. However, even with these differences, the functionality, i.e. the observable behavior, of the diagrams can be the same. This makes "back-translation" of the generated computer code, for example C code, to function block diagrams within the engineering database very complicated.

The object of the present invention therefore is to provide a method which reliably and accurately allows determining the equivalence of two different pieces of generated FBD source code.

Disclosure of Invention

In a first aspect, the invention relates to a method for checking the functional equivalence of two pieces of software for control systems, in particular control systems based on programmable logic controllers, engineered from function block diagrams with a plurality of interconnected function blocks, the method comprising the following steps:

a) generating a first instance of compilable source code from a first plurality of function block diagrams by virtue of a first code generator obeying to a first set of sequentialization rules;

b) parsing the first instance of source code and identifying the set of all function block I/O ports and junction points (nodes) and the set of all connections between them (arcs) of the data flow;
c) using the information from step b) to reconstruct a first data flow graph of the first instance of source code in the upstream direction;
d) generating a second instance of compilable source code from a second plurality of function block diagrams by virtue of the same or a second code generator obeying a second set of sequentialization rules;
e) parsing the second instance of source code and identifying the set of all function block I/O ports and junction points (nodes) and the set of all connections between them (arcs) of the data flow;
f) using the information from step e) to reconstruct a second data flow graph of the second instance of source code in the upstream direction;
g) comparing the first identified data flow graph and the second identified data flow graph with each other, node by node, and thereby checking if the transitive closure of input up to that node in the second instance of source code is the same as in the first instance of source code.

The invention is based on the consideration that a simple comparison of the differences of two source codes generated from FBD is not efficient due to possibility of many false positives due to freedom in the code generation, for example in the sequentialization of the function blocks used in the diagram. It would therefore be desirable to have a procedure which abstracts from these differences but maintains the essential information.

Applicant has recognized that a reliable and accurate comparison of two pieces of computer code can be achieved by generating data graphs from the code blocks which contain incoming and outgoing data. The method according to the present invention can reliably determine if parts of two I&C specifications are guaranteed to have the same behavior, even if they span multiple FBDs. It achieves this by parsing the generated (C or any other formally specified) code and traversing the data flow graph for each function block (FB) in the upstream direction. If all upstream FBs are of the same type, have the same parameters, and are linked

with other FBs or input signals by equivalent vertices, then they perform the same I&C function. In this way it is possible to prove that two compared sets of FBDs are functionally equivalent.

The method according to the present invention comprises transforming the generated code of FBDs, which is a directed multigraph, into a directed graph which can uniquely model the functionality implemented by the FBD. The representation of the graph is done in a way which allows a unique identification of each node and each vertex in the graph by including all relevant functional properties and the transitive closure of inputs up to each node. This allows a comparison of different FBDs to prove whether they implement the same functionality or not.

The method is a computer-implemented method which runs on a computer, preferably allowing a user to interact with it and preferably connected to a display device on which the user gets shown the result of the code comparison.

In a preferred embodiment, constructing the first data graph in the upstream direction comprises identifying all loops and replacing them with a placeholder such that an infinite recursion is stopped at the loops' delay nodes.

In another preferred embodiment, constructing the second data graph in the upstream direction comprises identifying all loops and replacing them with a placeholder such that an infinite recursion is stopped at the loops' delay nodes.

Preferably, a first and/or second plurality of function block diagrams by virtue of said respective first code generator is generated into source code in the C computer language.

Preferably the second set of sequentialization rules is identical to the first set of sequentialization rules.

Advantageously, the first and/or second data graph comprises nodes of the following types:

- function block nodes, representing the body of a function block;
- function block port nodes, representing input or output ports of function blocks;
- signal negator nodes, representing the negation of an analog or binary or other type of signal connected at a port of a function block; and
- signal nodes, representing cross-diagram signals or signals assigned to I/O modules.

Function block nodes preferably comprise as attributes, respectively, the function block type and functionally relevant engineered parameter values, and whereby function block port nodes comprise as attributes, respectively, the direction of the port (input or output for the function block) and a number identifying the port of the respective function block.

Preferably the method described above is employed for safety control system, for example in nuclear power plants.

Advantageously, the method described above is employed for a safety control system comprising several redundant divisions, each division executing at least one savety function specified in at least one function block diagram.

In a second aspect, the present invention relates to a non-transitory computer-readable storage device, coupled to a computer and having instructions stored thereon which, when executed on said computer, enable and/or cause the computer to perform operations for checking the functional equivalence of two pieces of software for control systems, the operations comprising an above described method.

In a third aspect the present invention relates to a non-transitory computer-readable storage medium containing instructions which, when loaded into memory of and executed on a computer, enable and/or cause the computer to perform

operations for checking the functional equivalence of two pieces of software for control systems, the operations comprising an above described method.

In yet another aspect, the present invention relates to a computer program product which is stored on a non-transitory computer-readable medium such as a flash disk, CD, DVD, hard disk and can be loaded directly into memory of a computer, comprises software code sections which when executed on the computer enables and/or causes said computer to perform a method described above.

In yet another aspect, the invention relates to a system comprising:

- at least one computing device;
- a non-transitory computer-readable storage device, coupled to said computing device and having instructions stored thereon which, when executed on said computer, enable and/or cause said computer to perform operations for checking the functional equivalence of two pieces of software for control systems, the operations comprising a method described above.

In yet another aspect, the invention relates to a computer comprising memory having stored thereon software code sections which when executed on said computer enable and/or cause the computer to perform a method described above.

The advantages of the invention are especially as follows. It is advantageous to work on generated code as opposed as to work on the graphical specification because in a regulated environment this is closer to the final product and avoids questions regarding the reliability of the translation step from graphical specification to code (the code generation). It is another advantage to be able to fully reconstruct the dataflow graph as opposed as to working FB by FB (as has been done by previous tools) because this captures all input data and hence makes a statement that "this piece of software does the same as that piece of software" 100% reliable.

Brief description of the drawings

Further features and advantages of the present invention shall become clearer from the following detailed description of some of its preferred embodiments, made with reference to the attached schematic drawings and given as an indication and not for limiting purposes.

In particular, the attached drawings are included to provide a further understanding of the invention and are incorporated in and constitute a part of this specification. The drawings together with the description explain the principles of the invention. In these drawings:

FIG. 1 shows a schematic diagram of protection functions in a safety control system;

FIG. 2 shows a diagram of function blocks and two different sequentializations of function blocks in the generated source code of this diagram:

FIG. 3 shows a comparison of generated FBD source codes using their graph representation;

FIG. 4 shows a first function block diagram comprising function blocks, connection lines and a signal negation;

FIG. 5 shows a visual representation of the graph modelling the function block diagram of FIG. 4;

FIG. 6 shows a second function block diagram comprising function blocks and connection lines; and

FIG. 7 shows a visual representation of the graph modelling the function block diagram of FIG. 6.

Detailed Description of the Invention

A typical structure 2 of protection functions is shown in FIG. 1. It consists of a set of I&C functions 6, 10, 14, 18 implemented in multiple redundancies in a control system for a nuclear power plant. A first of these functions 6 receives a signal from one or more sensors 26 which is then processed in a computational block 30. The calculated result is then passed through a safety network 34 to another computational block 38 in which it is processed into a signal for controlling one or more actuators 42. The other functions 10, 14, 18 are realized in the same manner.

In order to implement functionality with multiple redundancies, the corresponding function block diagrams for functions 6, 10, 14, 18 are implemented multiple times. In a typical protection system, certain diagrams have to be implemented up to four times to implement such redundant division even though the logic implemented by these diagrams is the same. For these function block diagrams, code is generated independently which results in four different source codes implementing equivalent functionality.

In a preferred embodiment of the method, function block diagrams are converted from an engineering database to ANSI C code by a code generator. In order to have unique identifiers in the generated source codes, two function block diagrams of one specification differ at least in naming of symbols. For example, symbols related to a function block diagram include the internal numerical ID of the diagram within their name. Additionally, identifiers in the generated code can also differ between two revisions of one diagram's source code. Reasons might be changes to the diagram itself, as well as changes to signals interconnecting diagrams. This is due to the fact that symbols in the generated codes may use internal database keys of the employed database that cannot be engineered. One example is specification-wide unique identifiers of cross-diagram signals.

The following samples show parts of function block diagram source codes, using numerical diagram IDs (number 99) and numerical signal IDs (number 1111).

```
void i_fd_99_Compute(void)
{
  static const fb_257_t locFb_1 = { /* FB: "SINK-2", Type: "SINK" */
  fd_99_FbExtIdents.id_257, /* Page: 1, Position: 3B */
  &i_fd_99_CSignals.asZ_1111, /* Source: Signal "IO READ A021" */
  &fd_99_USignals.bs_257_2,
...
}
```

Another point where two functionally identical diagrams can differ in their source code is the sequentialization of function block calls. The same diagram can result in different permutations of function block sequentialization while still being functionally equivalent (i.e. have the same observable behavior). For example, in FIG. 2 a very simple diagram 50 is shown with three different function calls A, B, and C which can, when converted into source code by a code generator 56, result in two sequentializations 60, 64 which both correctly implement the diagram. In sequentialization 60, first function A is called, then function B is called and then function C is called. In sequentialization 64, first function B is called, then function A is called and then function C is called.

The following samples show parts of function block diagram source code containing IDs depending on the sequentialization of the function blocks.

```
static const fb_123_t locFb_0 = {…};
static const fb_321_t locFb_1 = {…};
```

All sequentializations are considered to be functionally equivalent if they comply with a set of given sequentialization rules. The code generator used in the present embodiment generates source code for function block diagrams which adhere to these rules. The exchange of data between function blocks within one function block diagram is strictly defined and uses dedicated objects for storing this data.

Equivalence checking of function block diagrams can be used as a formal procedure of verifying that the code generated from function block diagrams behaves exactly as specified in terms of a circuitry on function block diagrams for

any sequence of input signals or a formal procedure of verifying that one set of code generated from function block diagrams behaves exactly as another set.

In order to compare function block diagrams based on the dataflow graphs they implement, the respective code is transformed into such a directed graph.

In FIG. 3, a first set of three pieces of source code 70, 74, 78 generated from function block diagrams in a transformation step 82 is transformed into a first data flow graph 86. A second set of three pieces of source code 90, 94, 98 generated from function block diagram is in a transformation step 102 transformed into a second data flow graph 106. In a comparing step 110, the data flow graphs 86, 106 are compared. If they are identical, the equivalence of the group of source codes 70, 74, 78 and the group of source codes 90, 94, 98 is asserted.

When comparing two directed graphs, by definition the graphs are equivalent if the two sets V (vertices) and A (nodes) are the same in both graphs. Such a comparison requires a unique identification of each node. With the invented method the nodes can be uniquelly idenfied by adding the nodes type, functionally relevant properties of the node and the transitive closure of all node inputs to the identification of each node.

A graph describing the logics of a function block diagram preferably consists of the following node types.
• function block nodes, representing the "body" of a function block;
• function block port nodes, representing input or output ports of function blocks;
• signal negator nodes, representing the negation of an analog or binary or other type of signal connected at a port of a function block; and
• signal nodes, representing cross-diagram signals or signals assigned to I/O modules.

Other attributes of function block diagrams which are relevant for the implemented functionality are modelled as properties of the graph's nodes. The following attributes are preferably taken into account.

For function block nodes:

- the function block's type;
- functionally relevant engineered parameter values.

For function block port nodes:

- the direction of the port (input or output),
- a identification of the port of the respective function block.

The dataflow between the engineered function blocks, function block diagrams and/or I/O modules is modelled by the directed edges which connect the nodes of the graph.

By creating a dataflow graph using the above mentioned nodes, their properties and interconnections of a function block diagram, the functionality implemented by the function block diagram can be modelled and the resulting graph can be compared to other graphs by comparing the contained nodes and edges

FIG. 4 shows a very simple function block diagram 120 containing four function blocks 124, 128, 132, 136, three connection lines 140, 144, 148 and one signal negation 150 at the upper input of the block 132 which is an ADD block, adding the negated signal transmitted from block 124 via connection line 140 and the signal transmitted from block 128 via connection line 144.

A visual representation of a graph modelling this function block diagram 120 is shown in FIG. 5. The boxes 160, 164, 168, 172 only mark the function block boundaries and do not model any nodes within the graph. The box 176 represents the signal negation 150 shown in FIG. 4.

FIG. 6 shows another simple diagram 196, implementing a slightly different logic. It contains four function blocks 190, 194,198, 202, three connection lines 206, 210, 214 but is missing the signal negation at the input of block 198 which is an ADD block. The missing signal negation is the only difference between diagrams 120 and 196.

The corresponding graph is displayed in FIG. 7. The boxes 220, 224, 228, 232 only mark the function block boundaries and do not model any nodes within the graph. The identification of the nodes contained within the graphs has to include the above named properties as well as the transitive closure of inputs of the nodes. The following example shows a more formal representation of the nodes shown in FIG. 7 including hierarchical information about the nodes inputs. For easier reading the example does not contain any other functional properties within the node identifiers.

- A-CONGEN()
- Out1(A-CONGEN())
- A-CONGEN()
- Out1(A-CONGEN())
- In2(Out1(A-CONGEN()))
- In3(Out1(A-CONGEN()))
- ADD(In2(Out1(A-CONGEN()))),In3(Out1(A-CONGEN())))
- Out1(ADD(In2(Out1(A-CONGEN()))),In3(Out1(A-CONGEN()))))
- In1(Out1(ADD(In2(Out1(A-CONGEN()))),In3(Out1(A-CONGEN()))))
- SINK(In1(Out1(ADD(In2(Out1(A-CONGEN()))),In3(Out1(ACONGEN()))))))

Combining such a representation of nodes with all relevant functional properties allows a unique identification of each node regarding the functionality implemented using this node.

Note: Looking at the example above, one can see that two functional equivalent nodes might have the same identifier, which is absolutely correct. For the example this is true for the two A-CONGEN function blocks and their output signals. In case they use different parameterization, the identification would differ due to these functional parameters.

When comparing the two graphs of FIGs 5 and 7, the set of contained nodes differs by the missing negator node. Additionally, the set of contained edges differs too, due to the missing node. The equivalence check based on the method according to the present invention preferably does not consider attributes that have no impact on the behavior of a function block diagram, such as:

• position and order the function blocks are drawn on the function block diagram,

• how connection lines are drawn (routing, including connectors across diagram pages), or
• position of cross-diagram signals in the diagram's input and output area.
Those are purely graphical attributes used for visual display of the function block diagrams and have no effect on the generated code's structure.

Using this method, each instance of function block diagrams can be transformed into a unique dataflow based graph using the functional relevant attributes listed above. By using only functional relevant attributes of function block diagrams for this transformation, the resulting graph representations can be compared. This yields a proof that two function block diagrams are functionally equivalent if their graph representations are identical.

When comparing multiple diagrams which are interconnected using signals, the data flow graphs are preferably analyzed across diagram boundaries. This means that differences in the interconnection of diagrams are detected, too. When evaluating the interconnections between function block diagrams, it is checked whether the compared signals have an equivalent source and destination node

The implemented method preferably allows the user to define where each set of function block diagrams shall be read from. This enables the user to specify either one code generation directory or two separate code generation directories for the comparison. The method is preferably used for
• two sets of diagrams from one version of an application, for example for comparing diagrams between redundancies;
• two sets of diagrams from two versions of the same application, for example to check that only planned changes where introduced, or even
• two sets of diagrams from two different applications, for example to check that copied diagrams were introduced correctly.

Attributes marked as optionally, are only preferably processed when explicitly requested by the user when starting the comparison. By default these attributes are ignored.

Claims

1. A method for checking the functional equivalence of two pieces of software for control systems, in particular control systems based on programmable logic controllers, engineered from function block diagrams (120, 196) with a plurality of interconnected function blocks, the method comprising the following steps:

a) generating a first instance of compilable source code from a first plurality of function diagrams (70, 74, 78) by virtue of a first code generator obeying to a first set of sequentialization rules;

b) parsing said first instance of source code and identifying the set of all function block I/O ports and junction points (nodes) and the set of all connections between them (arcs) of the data flow;

c) using the information from step b) to reconstruct a first data flow graph (86) of said first instance of source code in the upstream direction;

d) generating a second instance of compilable source code from a second plurality of function block diagrams (90, 94, 98) by virtue of the same or a second code generator obeying a second set of sequentialization rules;

e) parsing said second instance of source code and identifying the set of all function block I/O ports and junction points (nodes) and the set of all connections between them (arcs) of the data flow;

f) using the information from step e) to reconstruct a second data flow graph (106) of said second instance of source code in the upstream direction;

g) comparing said first identified data flow graph (86) and said second identified data flow graph (106) with each other, node by node, and thereby checking if the transitive closure of input up to that node in said second instance of source code is the same as in said first instance of source code.

2. Method according to claim 1, whereby said second set of sequentialization rules is identical to said first set of sequentialization rules.

3. Method according to claim1 or 2, whereby constructing said first data flow graph (86) in the upstream direction comprises identifying all loops and replacing them with a placeholder such that an infinite recursion is stopped at the loops' delay nodes.

4. Method according to one of the claims 1 to 3, whereby constructing said second data flow graph (106) in the upstream direction comprises identifying all loops and replacing them with a placeholder such that an infinite recursion is stopped at the loops' delay nodes.

5. Method according to one of the claims 1 to 4, whereby an equivalence of said both pieces of software is determined by checking if for each node the transitive closure of input up to that node in said second instance of source code is the same as in said first instance of source code.

6. Method according to one of the claims 1 to 5, whereby said first and/or second plurality of function block diagrams by virtue of said respective first code generator is generated into source code written in the C programming language.

7. Method according to one of the claims 1 to 6, whereby said first and/or said second plurality of function block diagram source codes (70, 74, 78, 90, 94, 98) comprises at least one element of the group: delay block, non-delay block, negator, and whereby said first and/or said second sequentialization complies with defined sequentialization rules.

8. Method according to one of the claims 1 to 7, whereby said first and/or second data flow graph (86, 106) comprises nodes of the following types:

- function block nodes, representing the body of a function block;

- function block port nodes, representing input or output ports of function blocks;
- signal negator nodes, representing the negation of an analog or binary or another type of signal connected at a port of a function block; and
- signal nodes, representing cross-diagram signals or signals assigned to I/O modules.

9. Method according to claim 8, whereby function block nodes comprise as attributes, respectively, the function block type and functionally relevant engineered parameter values, and whereby function block port nodes comprise as attributes, respectively, the direction of the port (input or output for the function block) and an identification of the port of the respective function block.

10. Method according to one of the preceding claims, which is employed for a safety control system in a nuclear power plant.

11. Method according to one of the preceding claims, which is employed for a safety control system comprising several redundant divisions, each division executing at least one savety function specified in at least one function block diagram.

12. A non-transitory computer-readable storage device, coupled to a computer and having instructions stored thereon which, when executed on said computer, enable and/or cause said computer to perform operations for checking the functional equivalence of two pieces of software for control systems, the operations comprising a method according to one of the claims 1 to 11.

13. A non-transitory computer-readable storage medium containing instructions which, when loaded into memory of and executed on a computer, enable and/or cause the computer to perform operations for checking the functional equivalence of two pieces of software for control systems, the operations comprising a method according to one of the claims 1 to 11.

14. A computer program product, which is stored on a non-transitory computer-readable medium and can be loaded directly into memory of a computer, comprises software code sections which when executed on the computer enables and/or causes said computer to perform a method according to one of the claims 1 to 11.

15. A computer comprising memory having stored thereon software code sections which when executed on said computer enable and/or cause said computer to perform a method according to one of the claims 1 to 11.

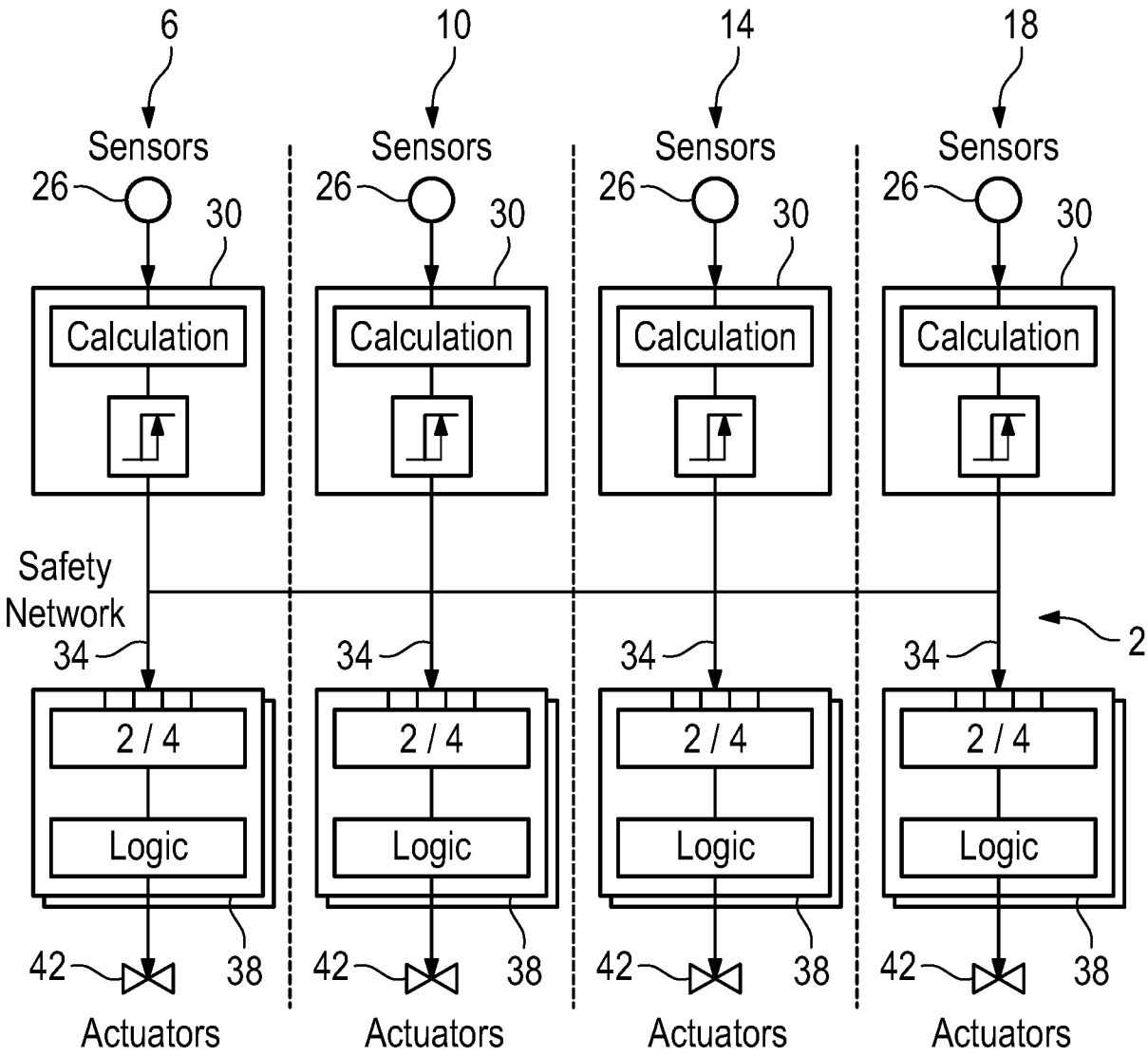

6
10
14
18
Sensors
Sensors
Sensors
Sensors
26
30
Calculation
Safety
Network
34
2
2 / 4
Logic
42
38
Actuators

Fig. 1

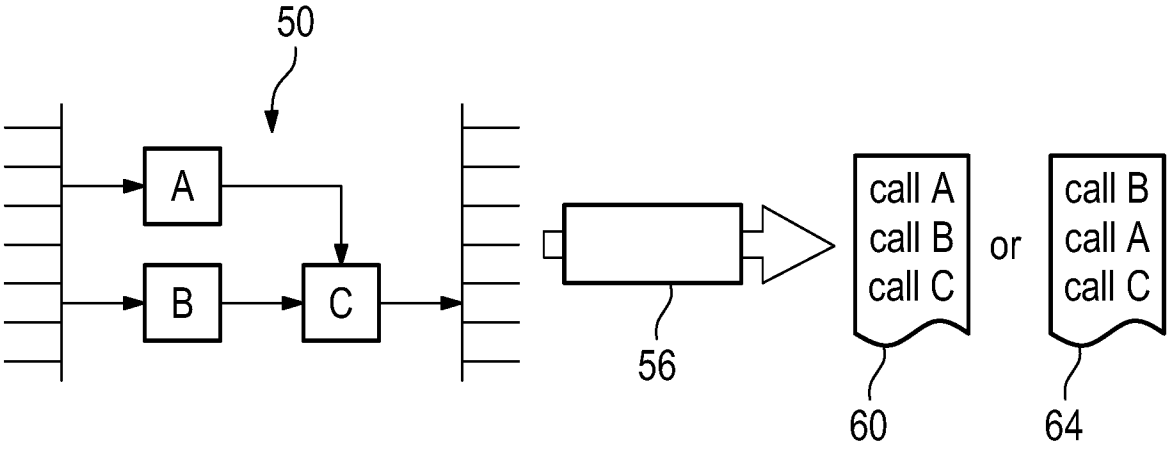

50
A
B
C
56
call A
call B
call C
60
or
call B
call A
call C
64

Fig. 2

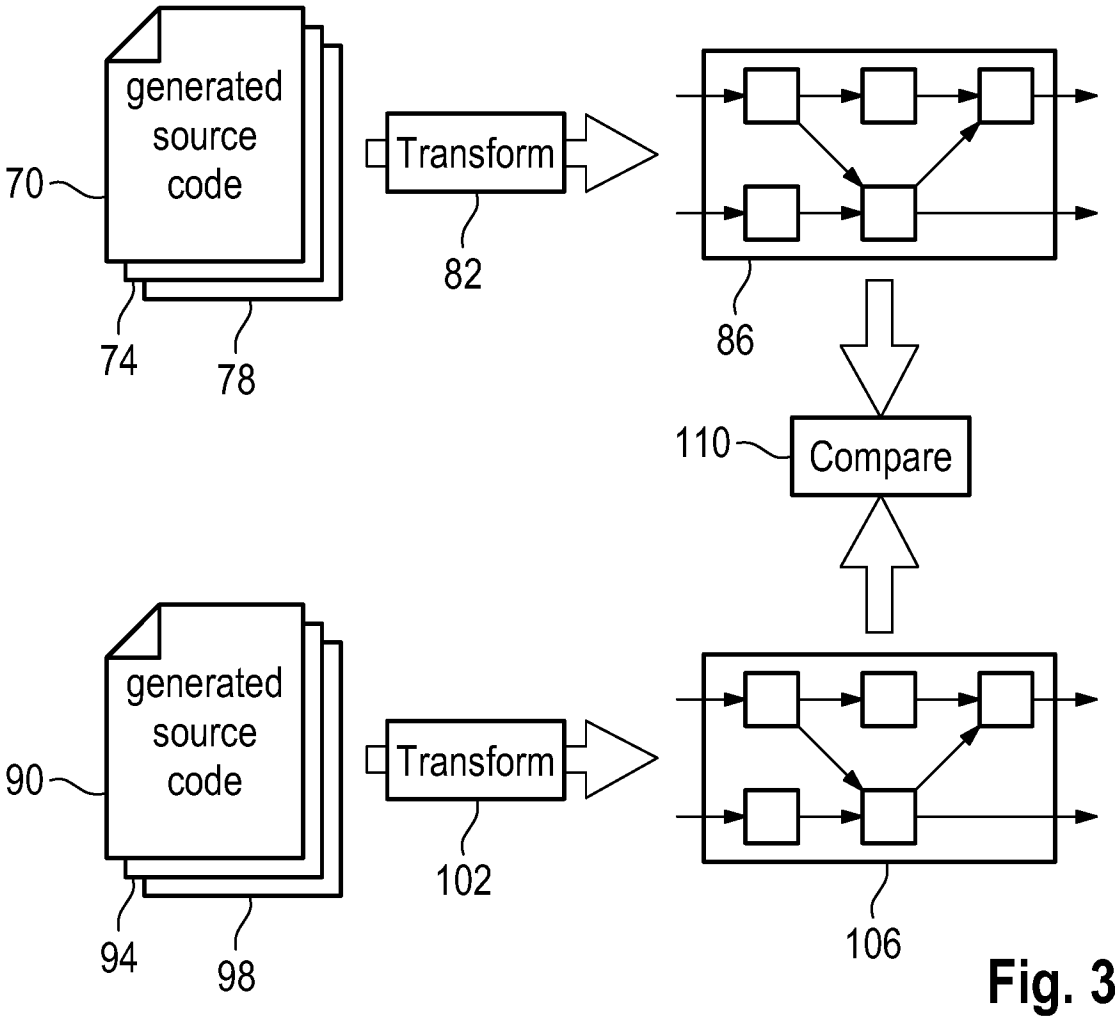

generated source code
70
74
78
Transform
82
86
110
Compare
generated source code
90
94
98
Transform
102
106

Fig. 3

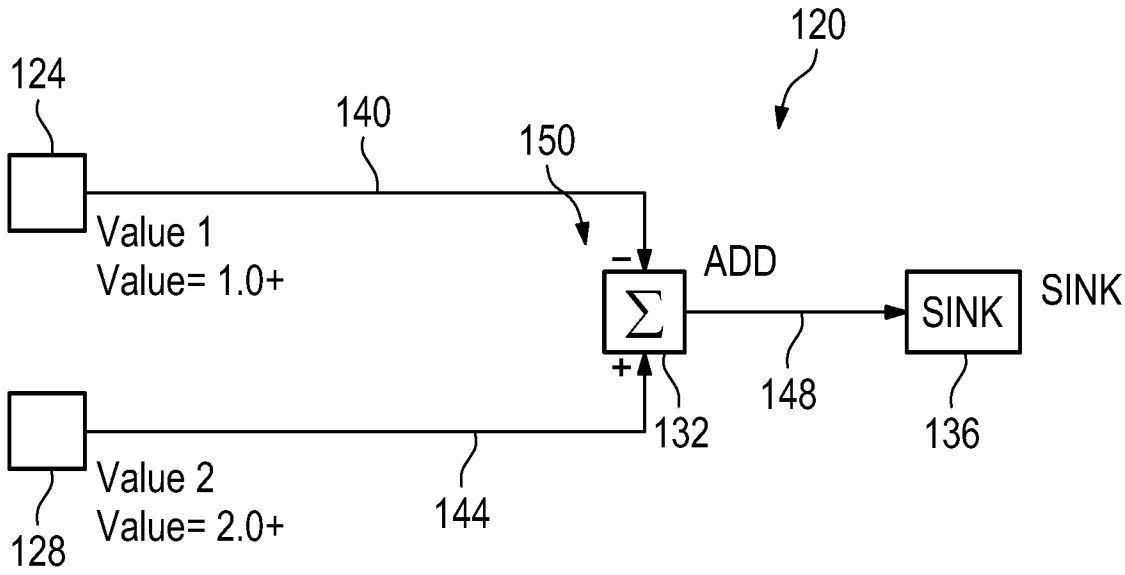

120
124
140
150
Value 1
Value= 1.0+
−
ADD
Σ
SINK
SINK
+
148
132
136
Value 2
Value= 2.0+
144
128

Fig. 4

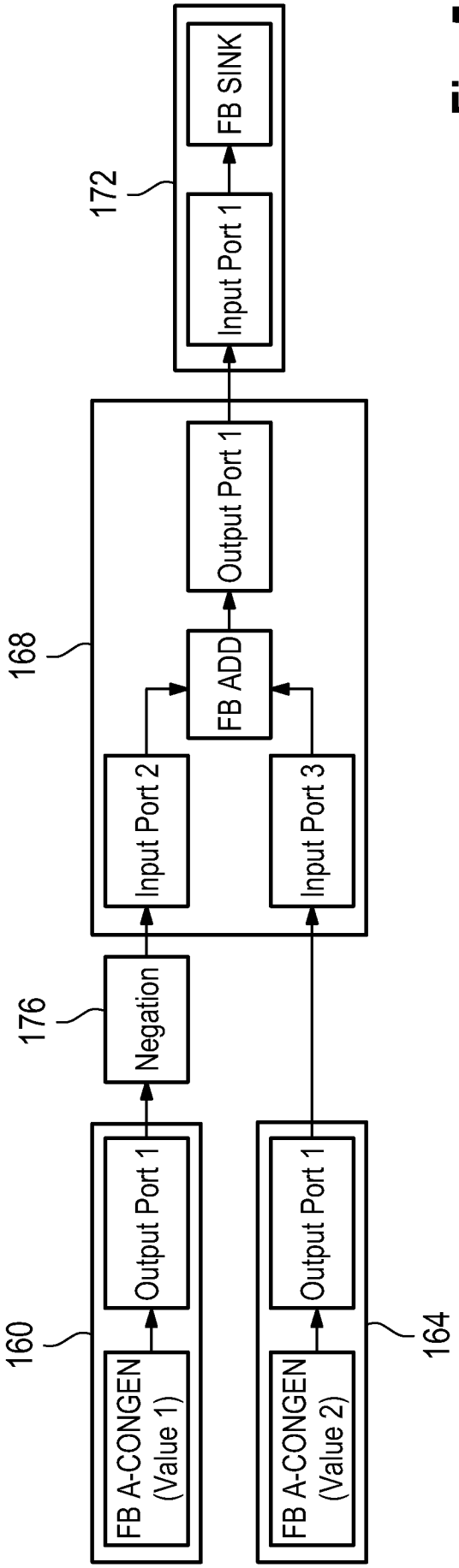

FB A-CONGEN (Value 1)
Output Port 1
160
Negation
176
Input Port 2
Input Port 3
FB ADD
Output Port 1
168
Input Port 1
FB SINK
172
FB A-CONGEN (Value 2)
Output Port 1
164

Fig. 5

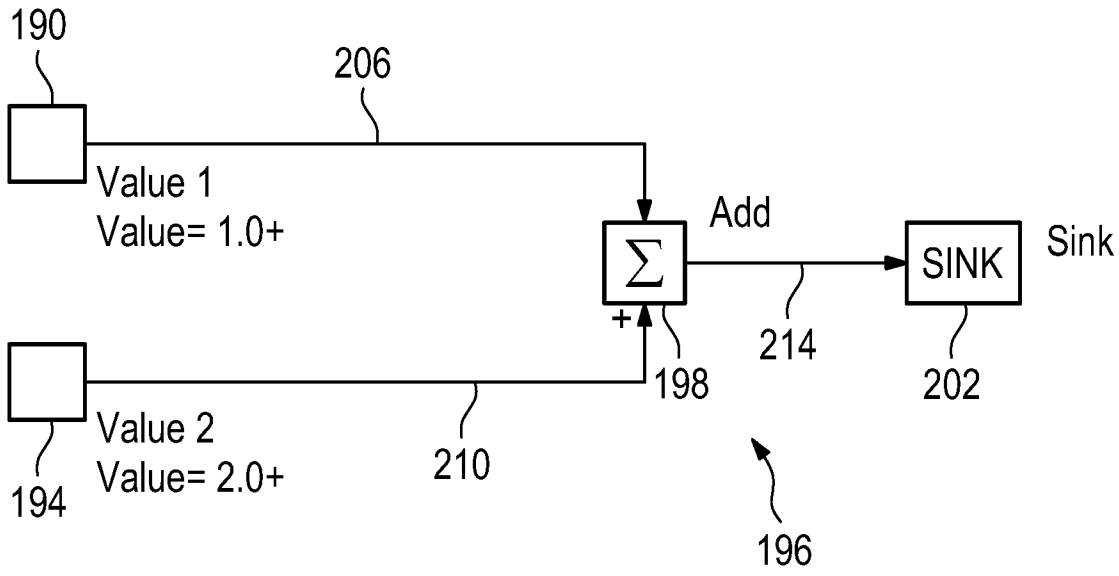

190
206
Value 1
Value= 1.0+
Add
Σ
SINK
Sink
+
214
198
202
Value 2
Value= 2.0+
210
194
196

Fig. 6

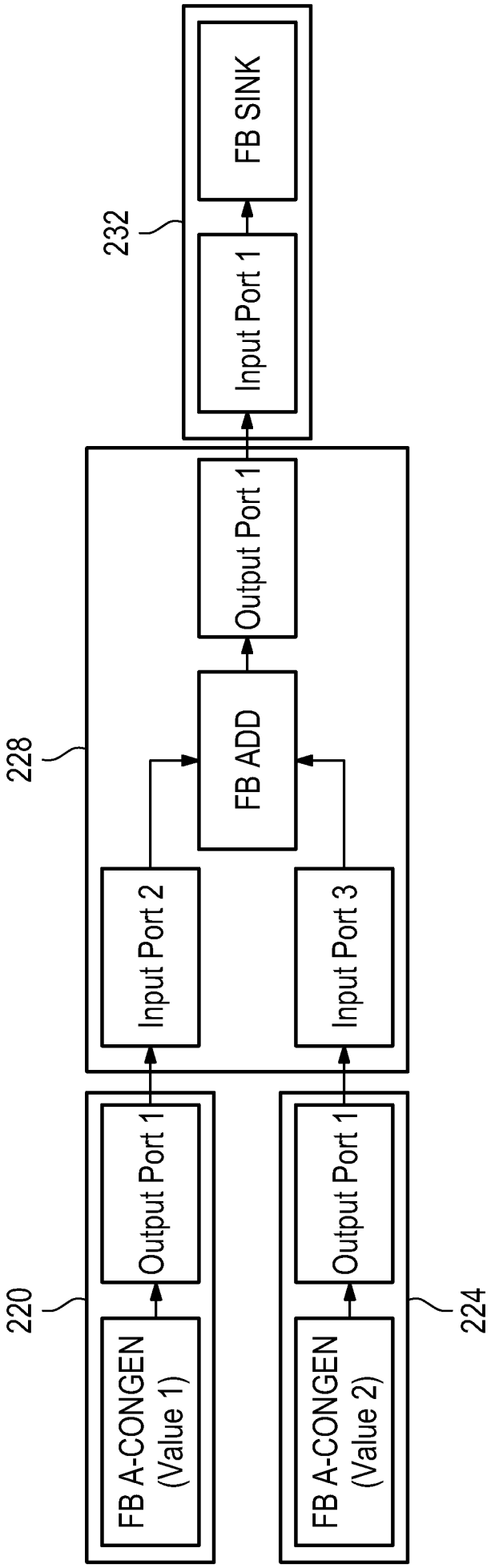

FB SINK
Input Port 1
232
Output Port 1
FB ADD
228
Input Port 2
Input Port 3
Output Port 1
Output Port 1
220
224
FB A-CONGEN (Value 1)
FB A-CONGEN (Value 2)

Fig. 7

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2016/059479

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F11/36 G06F17/50
ADD.

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)
G06F

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 practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 2010/053603 A1 (INTRINSITY INC [US]; SHEELEY NATHAN FRANCIS [US]; NODINE MARK H [US];) 14 May 2010 (2010-05-14) paragraphs [0013], [0038] - [0064]	1-15
X	US 2011/093824 A1 (BAUMGARTNER JASON R [US] ET AL) 21 April 2011 (2011-04-21) claim 1; figure 4	1-15
X	US 2009/300563 A1 (MOON IN-HO [US]) 3 December 2009 (2009-12-03) claim 1; figure 11	1-15

☐ 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 application or patent 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
11 July 2016	18/07/2016

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	Renault, Sophie

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2016/059479

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2010053603 A1	14-05-2010	US 2011214096 A1	01-09-2011
		WO 2010053603 A1	14-05-2010
US 2011093824 A1	21-04-2011	US 2011093824 A1	21-04-2011
		US 2013037998 A1	14-02-2013
US 2009300563 A1	03-12-2009	NONE	