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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 December 2012 (06.12.2012)

WIPO | PCT

(10) International Publication Number
WO 2012/164541 A1

(51) International Patent Classification:
H03K 3/037 (2006.01) *G06F 11/00* (2006.01)

(21) International Application Number:
PCT/IB2012/052783

(22) International Filing Date:
1 June 2012 (01.06.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
TO2011A000485 3 June 2011 (03.06.2011) IT

(71) Applicant *(for all designated States except US)*: **POLITECNICO DI TORINO** [IT/IT]; Corso Duca Degli Abruzzi 24, I-10129 Torino (to) (IT).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **LAVAGNO, Luciano** [IT/IT]; c/o POLITECNICO DI TORINO, Corso Duca Degli Abruzzi 24, I-10129 Torino (IT). **CANNIZZARO, Marco** [IT/IT]; c/o POLITECNICO DI TORINO, Corso Duca Degli Abruzzi 24, I-10129 Torino (IT).

(74) Agents: **BORSANO, Corrado** et al.; C/o Metroconsult S.r.l., Foro Buonaparte No. 51, I-20121 Milano (IT).

(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, 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, 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, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, 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, 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, ML, MR, NE, SN, TD, TG).

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

(54) **Title**: METHOD AND CIRCUIT FOR SOLVING METASTABILITY CONDITIONS AND RECOVERING SIGNAL ERRORS IN DIGITAL INTEGRATED CIRCUITS

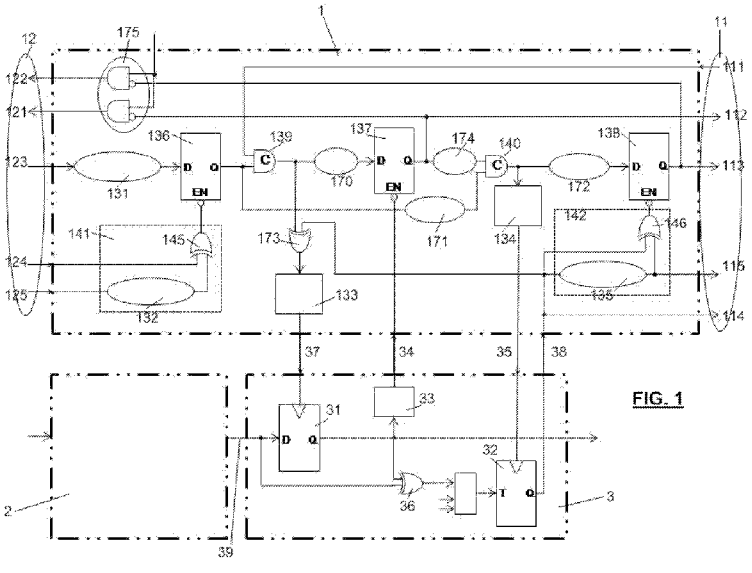

FIG. 1

(57) **Abstract**: It is described a method for solving meta-stability conditions and recovering signal errors in a digital integrated circuit, comprising the following steps: detecting meta- stability in the digital integrated circuit; using the information on said meta-stability to temporarily stop the clock generation for the digital integrated circuit; restart the clock generation for the digital integrated circuit when meta-stability has been resolved. The method further comprises a step of recovering signal errors including detecting errors, and repeating the computation performed by the digital integrated circuit until the error conditions have been resolved. It is also described a number of variants of circuit embodiments of the method.

WO 2012/164541 A1

METHOD AND CIRCUIT FOR SOLVING METASTABILITY CONDITIONS AND RECOVERING SIGNAL ERRORS IN DIGITAL INTEGRATED CIRCUITS

DESCRIPTION

Field of the invention

The present invention relates to a method and circuit for solving metastability conditions and recovering signal errors in digital integrated circuits.

Description of the prior art

Process and operating condition variability creates a huge problem for current and future digital integrated circuits, namely it forces them to operate at a speed, voltage and hence power and energy consumption which is very far from the optimum. The reason is that it is necessary to ensure, at design time, that each individual circuit will work, by assuming always *worst-case conditions*, no matter what its speed would have been, due to its fabrication process and current operating conditions. With modern fabrication technology this routinely means operating at a frequency which is less than half of what it could be if a circuit were allowed to run at its own optimum speed, and, as far as power/energy is concerned, this means operating at 4-10X the optimum power level.

This huge gap is currently addressed, especially in terms of optimizing power consumption, by using:

- "Binning", a solution for recovering some of the performance that is lost, namely that due to process variability, by separating ICs into various performance bins and selling them at different prices. This is applicable only to processors, memories, FPGAs and graphics chips, due to their market sizes.
- Adaptive voltage scaling techniques, which *estimate* the current performance of the circuit, due to both fabrication and current operating conditions, and adapt the voltage supply to meet the performance requirements at a reduced power and energy consumption level. This is routinely applied to portable electronics, such as cell phones. However, it still requires a large margin due to the fact that performance is only *estimated*, by using a chain of gates that somehow replicates the delay of the critical paths of the circuit. It is estimated that this leaves about 10-20% of performance unexploited (this percentage is growing with every technology generation) and 2X of power and energy.

A technique is known, as disclosed in US-7,337,356, US-7,320,091, US-7,162,661, which instead of binning or estimating the performance of the circuit, proposes to actively strive to run the circuit at the lowest supply voltage or highest performance (depending on the goal of the IC, performance for CPUs and GPUs and power for portable electronics), and recover from the errors that may occur as a consequence. The reason for errors to occur is that if a circuit is operated near its frequency/supply voltage limit, some flip-flops within it may memorize the incorrect value from the data-path when they receive a clock edge. Thus this known technique provides for latching the same value half (or some fraction of) a clock cycle later, and comparing the two values. If they are identical, it concludes that no error occurred, and lets the circuit proceed. If a mismatch is detected, it stops the circuit (which has already started computing based on the wrong value) and restarts it from the correct value which has been memorized later. In this way, no incorrect value is ever propagated, and the circuit can operate virtually without margins.

The main problem with the use of this known technique stems from the fact that the flip-flop may not only memorize the wrong value if it is clocked too soon. It may also *become meta-stable*. Meta-stability results in a non-digital output being produced, and remaining for a non-deterministic amount of time. Furthermore, it is unavoidable, and causes the circuit to seriously malfunction, because a non-digital output may be interpreted as two different values by different gates, and hence may cause the function and the state of the circuit to become completely incorrect, in an unrecoverable manner. By this known technique this problem is handled by detecting meta-stability, and by re-starting the circuit from a safely memorized state before the first meta-stability error occurs. This is possible only with processors, because they already contain that logic (e.g. to handle arithmetic errors or page faults). Other circuits would require very expensive check-pointing logic to be added, and hence this known approach is not amenable for them. Moreover, even with architectures such as processors, that already include the check-pointing logic or to which the check-pointing logic can be added, this known mechanism may not detect metastable failures which last for a short time, but may still result in incorrect data propagation beyond the so called “razor flops”.

Generally the main problem with meta-stability is not its detection, which can be done in a short amount of time (for example with a known circuit which checks if two signals are at a very close voltage), but rather the *sampling of the metastability error signal in a*

synchronous circuit, since that signal may have a *falling transition* (meaning that meta-stability has been resolved) an arbitrary amount of time after the rising transition. This can lead to both very short pulses and very long pulses on the metastability detection signal. The duration is a non-deterministic variable, the distribution of which depends on the circuit speed. One commonly assumes that the probability of incorrectly sampling a transitioning signal (and hence propagating meta-stability to the error detecting logic itself) is negligible if one samples the meta-stable signal 2-3 clock cycles after it has been memorized (by using 2-3 flip-flops which act as synchronizers). Hence the check-pointing mechanism above requires memorizing the state of the circuit for 2-3 clock cycles, which would cause a huge area consumption and power cost.

SUMMARY OF THE INVENTION

It is the main object of the present invention to provide a method and circuit for solving metastability conditions and recovering signal errors in digital integrated circuits, which overcomes all the drawbacks described above.

The basic idea of the present invention is to deterministically detect meta-stability conditions, use this information to temporarily stop the clock generation for the circuit (this can be done safely), and then restart it when meta-stability has been resolved.

Preferably an additional idea is to extend the features of the invention to the detection of error conditions, using this information to repeat the operations affected by error until the error has been resolved.

The approach of the present invention, for efficient detection and correction of circuit timing errors, is based on the speculative execution of the combinational logic in order to increase the throughput - or to lower the supply voltage - of the system. The speedup is achieved by increasing the clock frequency. In addition the clock can be stopped every time either one (or more) of the registers enters the metastable condition, or an error must be corrected, or both.

This approach can be applied in a non-limited way both to asynchronous circuits (for example with a 2-phase or a 4-phase protocol), and to the generation of synchronous local clocks in a Globally-Asynchronous Locally-Synchronous (GALS) architecture.

Some embodiments of the present invention provide for using *asynchronous circuits*, instead of the standard synchronous ones, because they allow to *stop the clock while meta-stability resolves*, without ever needing to sample the meta-stability detection signal with a synchronous unstoppable clock. This allows to *limit meta-stability to a*

single flip-flop, and hence it is possible to implement the invention by a simple circuit without expensive check-pointing registers or logic circuit modifications, only by changing locally the flip flops with "safe flip-flops" and by using one or more locally generated stoppable clock generators. The data-path of the circuit (combinational logic) and its overall architecture is not changed at all.

It is an object of the present invention a method for solving meta-stability conditions in a digital integrated circuit, comprising the following steps:

- detecting meta-stability conditions in the digital integrated circuit;
- using the information on said meta-stability conditions to temporarily stop the clock generation for the digital integrated circuit;
- restarting the clock generation for the digital integrated circuit when meta-stability conditions have been resolved.

Preferably the method also provides for a further step of signal error recovering, comprising: detecting error conditions in the digital integrated circuit; repeating the computation performed by the digital integrated circuit until the error conditions have been resolved.

Preferably the method, in case said digital integrated circuit comprises a number of interconnected asynchronous blocks, also comprises the steps of: exchanging the error condition information between said interconnected asynchronous blocks; temporarily stopping and restarting the clock generation of an asynchronous block of said number of interconnected asynchronous blocks, based also on error condition information received from one or more upstream blocks of said number of interconnected asynchronous blocks.

Also preferably the step of exchanging error condition information between said interconnected asynchronous blocks in based on an handshake procedure.

Still preferably the method comprises exchanging, between upstream and downstream interconnected stages, acknowledgment and request signals to communicate that the downstream stage is ready to receive a new data, and the upstream stage outputs a new data, said data being of a speculative type if possibly affected by errors, or of a non-speculative type, if not affected by errors.

It is a further object of the invention a circuit for the implementation of the method.

These and further objects are achieved by means of a method and circuit as described in the attached claims which are considered as an integral part of the present description.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will become fully clear from the following detailed description, given by way of a mere exemplifying and non limiting example, to be read with reference to the attached drawing figures, wherein:

- Fig. 1 shows a circuit block diagram of a first embodiment of the invention applied to a 2-phase asynchronous pipeline;
- Fig. 2 shows a flow-chart of the operations of the circuit of Fig. 1;
- Fig. 3 shows a simplified block diagram of another embodiment of the invention having a balanced asynchronous multi-stage pipeline structure based on the structure of the first embodiment of Fig. 1;
- Fig. 4 shows a block diagram of another embodiment of the invention having a multiple input/output asynchronous pipeline structure based on that of Fig. 1;
- Fig. 5 shows a circuit block diagram of a further embodiment of the invention applied to a 4-phase asynchronous pipeline;
- Fig. 6 shows a circuit block diagram of a further embodiment of the invention applied to the generation of synchronous local clocks in a GALS architecture.

The same reference numerals and letters in the figures designate the same or functionally equivalent parts.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

A number of embodiments of the invention will be described using asynchronous circuits. A first embodiment is described based on a 2-phase asynchronous pipeline. Then a second embodiment is described, showing how the previous architecture can be easily extended in order to handle "fork and join" procedure for a number of interconnected asynchronous blocks in the circuit so that any block of the circuit can send and/or receive data to/from any other blocks. Then a further embodiment will be described showing how the architecture can be easily adapted for a 4-phase protocol.

A further embodiment of the invention will be described applied to the generation of synchronous local clocks in a Globally-Asynchronous Locally-Synchronous (GALS) architecture.

1. Circuit based on 2-phase asynchronous pipeline.

Figure 1 shows one stage of a first embodiment of the circuit with a 2-phase asynchronous pipeline. It is composed of three blocks, delimited by the dotted lines in the figure: a clock generator block 1 which is synchronized, through an improvement of

the known "handshake" procedure, with the clock generator blocks of the other stages, and generates the local clock; a combinatorial logic block 2 which elaborates the data input; and a storage block 3 which stores the data output and detects if it is metastable or corrupted.

It is worth noting that the invention is not limited to "linear pipelines", which are used in the embodiment examples specifically described here only for the sake of illustration. It can be used, with the aid of standard asynchronous design techniques, such as the use of "fork" and "join" components, to any asynchronous circuit composed by an arbitrary interconnection of 2-phase or 4-phase controllers.

In the following the term "speculative data" will be used to identify the data stored in the main register which is not metastable but which may be incorrect; while the term "non-speculative data" will be used to identify the data stored in the main register which is not metastable and is also correct.

Storage block 3

The storage block 3 contains the *main* flip-flop 31 and the *error detecting* flip-flop 32. The *main* flip-flop 31stores the speculative data coming from the combinatorial logic 2 and, if an error is detected, then it is updated with the new correct data. An *error detecting* flip-flop 32 is introduced, according to one aspect of the invention, which receives the information of whether the main flip-flop 31 has stored a corrupted data value or not.

If the setup (or hold) time of the *main* flip-flop are violated, a metastability detection block 33, attached at the output Q of the *main* flip-flop 31, detects that the output voltage does not resolve to a definite high or low voltage, and it quickly (within a deterministically bounded amount of time) communicates a metastability condition to the clock generator block 1, through connection 34. The metastability detection block 33 can be made in a known way, i.e. it outputs a logic level if the input is a true logic "0" or "1", and outputs the opposite logic level if the input signal is in a metastable voltage condition.

The metastability detection block 33 also receives the outputs of all the main flip-flops parallel to 31 present in block 3 and not shown in Fig. 1.

The error is detected by the comparison between the data stored in the main flip-flop 31 (output Q) and the data which has arrived later at its input (D). The error detecting T-type flip-flop 32 is enabled by a delayed asynchronous clock received from block 1, line

35. The *T* input of the error detecting flip-flop 32 is the output of the tree of OR2 gates (36 shown in Fig. 1) which collects all outputs of the XOR2 gates coming from the number of main flip-flops parallel to 31 present in block 3.

Each XOR2 gate 36 receives the *D* and *Q* signals of the relating main flip-flop as inputs. This means that it checks if the input of the main flip-flop 31 has changed after the main clock rising transition received at input 37. The error detecting flip-flop 32 has a level-encoded (in this example '1' means 'error detected' and '0' means 'no error detected') error signal at input T and, if an error is detected, it generates a transition-encoded (any transition in the wire means 'error detected') error signal at the Q output 38. Since the circuit uses a transition signaling protocol, no reset phase is needed.

Preferably the error-detecting and the MD functional blocks need to be added only for those main flip-flops of the storage block 3 which can be affected by incorrect data or become metastable due to adaptive timing.

Clock generator 1: input and output interface.

The clock generator block 1 sends the two clocks (*main* clock 37 and *error detecting clock 35*) to the storage block 3, and receives the *metastability* signal 34 and the *error* signal 38 from the storage block 3. Figure 1 also shows all input and output signals which can be used to synchronize the clock generator to other asynchronous stages. However, depending on some conditions discussed below, some input/output signals may not be used. If one or more input/output signals are not used in the clock generator block, the circuit can be optimized by removing the corresponding circuitry. The following complete list of the interface signals is divided in two parts: right interface 11 and left interface 12.

Right interface 11.

The right interface is used to synchronize with the next stage in the chain (bundle 11), and comprises the following signals.

- an ***Acknowledge right interface:*** it is a one-wire signal input 111 used by the *next* stage to communicate that it is ready to receive a new data input.
- a ***Request right interface:*** it is composed of two signals: (i) a speculative request output 112, and (ii) a non-speculative (correct) request output 113. Each of them is a one-wire signal output used to communicate to the next stage that the data output contains a speculative data in the (i) case, or a correct data in the (ii) case. Only one of them is sent to the next stage, depending on the kind of clock generator of the next

stage and on the criticality of the combinatorial logic block 2 of the present and next stage (see next section for more details). Using the speculative request right signal output 112 requires that the error right interface (described below) must be connected to the error left interface (described below) of the next stage: therefore this can be done only if the next stage is controlled by an equivalent architecture as that described here. In this case, an error in the speculative data must be detected and corrected before the speculative data arrives to the input interface of the storage block of the next stage. The cases in which this should be done are discussed below. Otherwise the non-speculative request right signal output 113 is used.

- an ***Error right interface:*** it is composed of two signals: (a) an error output 114, and (b) a valid output 115. With these signals, the clock generator 1 can communicate the following information to the next stage: (a) when an error has been detected in the data output, activating the error output 114, and (b) when the data output has been corrected, activating the valid output 115. This interface is connected to the next stage only if the speculative request signal output 112 is used.

Left interface

The left interface is used to synchronize with the previous stage in the chain (bundle 12).

- an ***Acknowledge left interface:*** it is composed of two signals: (i) a speculative acknowledge output 121 and (ii) a non-speculative acknowledge output 122. Both of them are a one-wire signal used to communicate to the previous stage that the present stage is ready to receive a new data input. In the (i) case, the stored last data output is still speculative; while in case (ii), the last stored data output is correct. Only one of them is connected to the previous stage , depending on the criticality of the combinatorial logic block 2 of the present stage (see below for more details). Using the speculative acknowledge left signal output 121 does not require any additional logic in the previous stage. In this case, an error in the speculative data stored in the main flip-flop 31 must be detected and corrected before a new data arrives to the input interface 39 of the storage block 3. The cases in which this should be done are discussed below.
- a ***Request left interface:*** it is a one-wire signal input 123 used by the previous stage to communicate that the data input contains a new data (speculative or not speculative).
- an ***Error left interface:*** it is composed of two signals: (a) an error input 124, and (b) a valid input 125. These signals are used by the clock generator 1 to stop its execution

when an error in the data input is signaled by (a) input 124, and to restart it when the input data is validated by (b) input 125. If these signals are not used, for example because the previous stage uses a different procedure based on non-speculative request, they must be set to the same constant logic value (either zero or one).

When to use speculative or non-speculative "handshake" procedures.

To summarize, the circuit can communicate to the previous stage (by the acknowledge left interface 121, 122) and the next stage (by the request right interface 112, 113) whether it has stored the speculative data (by the speculative signals 112, 121) or the valid data (by the non-speculative signals 113, 122).

In order to take advantage of the speculative execution, the circuit should always use the speculative "handshake" procedure, and therefore its speculative output signals.

However, in the following conditions, the non-speculative output signals should be used, in order to reduce the complexity of the hardware:

- as mentioned above, if the output of the storage block is sent to another domain designed with a different architecture with respect to the one of the present invention, *the non-speculative request right signal output 113 must be used* since the receiver of the data output does not have the error left interface required by the speculative request.
- if the combinatorial logic 2 controlled by the clock generator 1 is not critical, using the speculative signals (both the acknowledge left 121 and the request right output 112) is not useful, since this would not speed up the system (whose performance is determined by the slowest logic). Therefore, *the non-speculative request right 113 and non-speculative acknowledge left signal output 122 should be used* in order to reduce the complexity of the hardware. In this case, the speculative execution of the combinatorial logic is not required and, therefore, the stage can use a standard register without the metastability and error detection in the storage block.
- if the combinatorial logic controlled by the clock generator is critical but the combinatorial logic of the next stage is not critical, using the speculative request output 112 is not useful since this would not speed up the system. Therefore, *the non-speculative request right signal output 113 should be used* in order to reduce the complexity of the hardware.

In all three cases above, the error right interface should not be used and the next stage should not need the error left interface, or the signals 124, 125 should be set to the non-active value.

It is worth noting that the information on meta-stability condition is not propagated through the stages of the pipeline, as the data, either speculative or non-speculative, that the other stages receive, following to a meta-stability detection and recovering in a given stage, will be stable anyway. In fact the detection of a meta-stability condition in a stage causes the temporary stop of the clock and handshake generation in that stage till the stable condition is recovered.

Clock generator: internal operation

The operation of the clock generator block 1 is explained here below with reference to the flow-chart diagram shown in Figure 2, structured according to a "fork and join" known type, and also with reference to Fig. 1.

Each new "local clock cycle" transaction starts from the left side of the clock generator block 1, when the system is waiting for the request left input 123 (block 211). After that request arrives, the clock generator 1 starts the waiting time for the execution of the combinational logic. The delay computed by block 212 (represented as delay 131 in Fig. 1) () is determined by considering two aspects: the *COMB* delay which matches the delay of the critical path of the *COMB LOGIC* block 2, minus the $\Delta error$ time, which can be modified to control the performance improvement due to adaptive timing in the stage. The latter could (and should in order to improve performance and/or reduce power consumption by reducing the supply voltage at equal performance) be greater than zero.

After this delay, the clock generator 1 checks (block 213) if there has been an error in the input data by looking at the error left signal input 124: if there has not been any (NO), the procedure goes on through the fork, which starts in parallel block 217 and another fork starting at point 214; otherwise (YES) it waits (block 216) until the valid input signal 125 (block 215) has arrived, after the $COMB - \Delta error$ delay generated by circuit 132 (meaning that the combinational logic has elaborated the valid data input), and then proceeds through the same fork, activating block 217 and the other fork. Signal 124 and the output of circuit 132 are brought to an EXOR 145, whose output is used as a clock for D-latch 136.

Then, the top branch (block 217) of the fork consists of just the $\Delta error$ delay; this is necessary to allow the correct detection of a possible error in the data stored in the main register 31.

The second branch of the fork first waits for the acknowledge right input 111 (block

218); then it generates by the circuit 133 (block 219) the clock pulse 37 for the main register 31 and, in parallel, a third branch is executed.

The latter starts (block 220) with a delay in order to wait and allow the storage block 3 to generate the metastability signal 34, and then the clock generator checks (block 221) whether the main register 31 is in a metastable condition or not. If the data output is not stable (metastability YES), the clock generator remains in this state, and the clock is halted.

If the metastability is resolved (metastability NO), two operations are executed in parallel:

- in one branch the speculative acknowledge left 121 and request right 112 output signals are generated, through D-latch 137 (block 222);
- in the other branch (block 223) there is a delay (generated by circuit 174, Fig, 1, connected between the output of D-latch 137 and one input of C-element 140), which allows to wait for the error propagation and the stable data stored in the main flip-flop 31 to arrive at the error detecting flip-flop 32 input before the clock pulse.

After that, the join node requires the algorithm to wait until the two branches of the fork are completed, then two operations are executed in parallel:

- (1) the clock pulse 35 to the error detecting register is generated (block 224, circuit 134); and
- (2) after a delay (block 225) in order to allow the storage block 3 to detect a possible error, it checks (block 226) whether an error has been detected or not.

If there are no errors (NO) in the data stored (no transition occurs on output 38), through activation by the clock generator 3, the non-speculative acknowledge left 122 and the non-speculative request right 113 output signals are generated by the D-latch 138 (block 227).

If an error has been detected (YES) by the storage block (a transition occurs on output 38), the circuit waits for a second condition (from block 228) always starting from the error signal 38 (block 230), but delayed by the error correction time (circuit 135, block 228).

When the error signal arrives, the clock generator also sends a new clock pulse 37 (block 219) to the main register 31 in order to restore the correct data. After the join, the system generates the valid right output signal 115 (block 229), the non-speculative acknowledge left 122 and non-speculative request right 113 output signals (block 227).

Depending on which signals (speculative or non-speculative acknowledge and request output) are used, the loop in the flow-chart is closed by one of the two dotted lines in Figure 2, closing on inputs of respectively block 211 or 218.

Clock generator: implementation

In the implementation shown in Figure 1, all wait statements of the flow-chart diagram (Figure 2) correspond to a delay which can be implemented in a known way in the wire, for example with an inverter chain.

The "if" statements are implemented with a D-latch (blocks 136, 137, 138). They are used to quickly stall all the control flow and clock generation in the following three conditions:

(i) the error input 124 signals that the data input was corrupted;

(ii) the storage block signals (signal 34) that the main register is in a metastable condition; and

(iii) the storage block signals (signal 38) that the main register 31 has stored a corrupted data.

The (ii) condition has a backward loop meaning that the system stays in that state as long as the condition is true. The (i) and (ii) conditions are modeled as a join in a forward loop, meaning that the system must wait a new condition to escape this state.

The other two join operations are implemented with C-elements 139, 140, which are equivalent to the AND gate in combinational logic. C-element 139 receives at inputs the output of D- latch 136 and signal 111 and sends the output to D-latch 137, through a delay block 170 of a time equal to the MD time of block 33. C-element 140, as mentioned before, receives at the inputs the output of D-latch 137 through the delay circuit 174, and the output of D-latch 136 through a delay block 171 of Δerror time, and sends the output to the pulse generator 134, and to the input of D-latch 138 through a delay block 172 of a time equal to the error detection .

Pulse generator blocks 141, 142 are used to generate the enable signals of the two D-latches 136 and 138. Pulse generator 141 comprises the EXOR 145 receiving the signals 124 and 125, the latter delayed by element 132, and generating the enable signal for the D-latch 136. Pulse generator 142 comprises the EXOR 146 receiving the signal 38 (that is also signal 114) both directly and delayed by element 135, and generating the enable signal for the D-latch 138.

Pulse generator blocks 133, 134 are used to generate the two clocks (main 37 and error

detecting 35 respectively). Pulse generator 133 is fed by the output of an EXOR 173, on turn receiving at the inputs the output of C-element 139 and the signal 38. Pulse generator 134 is fed by the output of C-element 140.

At any transition of the input signal, Pulse generator blocks 133, 134 generate a pulse on the output signal. They can be easily implemented with an XOR2 gate with both inputs connected to the pulse generator input but one of them is delayed. The latter delay is equal to the width of the pulse generated in the XOR2 output.

The output of D-latch 138 (signal 113) is applied to an inverted input of an AND gate (in the circle 175 in Fig. 1), and the output of D-latch 137 (signal 112) is applied to an inverted input of another AND gate (in the circle 175 in Fig. 1). The outputs of the two AND gates are respectively signal 122 and 121. At the second input of the two AND gates a reset pulse is applied. When the reset signal is low, both the acknowledge left output signals 121 and 122 are at zero, preventing the previous stage from generating transitions on the clock signal; when the reset signal goes high, the circuit is enabled and the previous stage can start its execution.

Example: balanced multi-stage asynchronous pipeline.

A balanced multi-stage asynchronous pipeline is made of a number of cascaded stages as that of Fig. 1. In it all stages are critical, as they can be subject to metastability and error conditions. Therefore all of them should use the speculative request and acknowledge signal output. Figure 3 shows a balanced asynchronous pipeline, according to another aspect of the invention, where the input of the first stage and the output of the last stage are assumed to be connected to an environment which works without the architecture of the present invention.

Therefore, as explained above, the non-speculative request must be used at the output of the last stage of the pipeline, as well as in the first stage the request input signal is connected to a non-speculative request output signal coming from another control logic built without the architecture of the present invention, and, therefore, the error left interface is not used. In Figure 3, only three stages are shown but one can obviously insert as many stages as needed, keeping the first and last stages without the speculative request.

2. Multiple input/output Asynchronous stage

The architecture discussed above, according to the invention, works with only one pair of left and right acknowledge and request signals. This means that the clock generator

block can only be synchronized with one left stage (which sends the data to its input) and one right stage (which receives the data from its output). Except for what is described here, the rest of the circuit of Fig. 4 is the same as in Fig. 1, and will not be described.

According to another aspect of the invention, the clock generator can be extended to receive more than one request left and acknowledge right input. Figure 4 shows one stage of the architecture which can receive *m* request left input signals (*q* of which are not speculative) and *n* acknowledge right inputs.

All the request signals coming from the left interface are gathered together in a tree of C-elements (circuit block 41) and its single-wire output enters the clock generator as in the previous architecture. In the same way, all the acknowledge signals coming from the right interface enter a tree of C-elements (circuit block 42). The error correction of the left input data (which is performed by stopping the request wire with a D-latch and a pulse generator) needs to be replicated for any of the *m-q* speculative request inputs.

Since the clock generator receives *m* request left signals from *m* other stages, the same number of acknowledge left signals (both speculative 121 and not speculative 122) must be sent to all of them.

In the other right interface, *n* request right signals (both speculative 112 and not speculative 113) must be connected to all other stages which receive the data output and send back the acknowledge signal.

3. Safe Razor Asynchronous stage (4-phase)

Figure 5 shows a Safe Razor asynchronous controller which works with a 4-phase protocol. It is similar to the architecture discussed above, with reference to Figure 1, which works with a 2-phase protocol. The only difference is that the pulse generator blocks which generate the main clock (block 133 in Fig. 1) and the error clock (block 134 in Fig. 1) are not needed any more, since the evaluation phase in the handshake protocol will cause the clock rising transition, and the reset phase will cause the clock falling transition.

A pulse generator 51 is still needed by the error signal before it is connected to the EXOR gate (block 173 in Fig. 1) which generates the main clock, if (as in this example) the error signal uses a 2-phase protocol where each transition must generate a pulse in the main clock.

The 4-phase operation is safer (because it does not require pulse generators) but

potentially slower (and more power consuming, with a large number of clock generators, each driving only a few flip-flops) than 2-phase operation.

4. Example of Synchronous architecture

Now it is shown how the circuit architecture of the invention can be extended so that the main clock can be sent to a standard synchronous module. Figure 6 shows the implementation of a synchronous module, which can be used in a Globally-Asynchronous Locally-Synchronous (GALS) architecture. In this case, the clock generator block 1"'' does not communicate with other GALS modules nor with their clock generators, because the *synchronization is done in the data path*. The input and output data are always *non-speculative* (meaning that no speculative data is sent to or received from other modules).

Like in the asynchronous Safe Razor architecture, this synchronous version is composed by the same three blocks, namely a clock generator 1"'', a combinatorial logic 2"'', and a storage block 3"''. In Fig. 6 the elements identified by the same reference number as in Fig. 1, accomplish the same function.

The storage block 3"'' has a loop back 611 to the combinatorial logic block 2"'' (this is the standard structure for any synchronous circuit). Therefore the combinatorial logic block 2"'' has two groups of inputs: one 612 coming from the environment and the other one generated by the storage block 3"''.

The storage block 3"'' sends to the combinatorial logic the speculative data stored 611 after the metastability is resolved.

To avoid a metastable data being sent via the loop back to the combinatorial logic after generation of the *main_clock* pulse 37 , (which could affect the error detection and correction mechanism), another flip-flop, called *stable* flip-flop 613, is added in the loop, and outputs the loop back signal 611; a new *stable_clock* signal 614 (generated by the clock generator block) is connected to its clock input.

One more flip-flop, called *correct* flip-flop 615, is also added between the output of the main flip-flop 31 and the output 616 of the storage block going towards other GALS modules. The clock input of flip-flop 615 is connected to another *correct_clock* signal 617 generated by the clock generator block.

The clock generator block, as already discussed in the previous chapter, must have a mechanism to stop the clock if an error is detected in any data input, and restart it when the data has been corrected and processed by the combinatorial logic block. Therefore,

there is a D-latch and pulse generator (controlled by the internal error signal, since the speculative input is the internal data input) so that a new *main_clock* pulse 37 may be delayed if an error is detected by the storage block.

The speculative and non-speculative request right signals (112, 113) are used to generate the two new clocks (i.e. the stable clock 614 and the correct clock 617). These two signals (which are not used as right interface outputs) are sent to the storage block through two pulse generator blocks, respectively 618 and 619.

Since the data stored in the *stable* flip-flop 613 must be updated if an error occurs, an XOR gate 620 is added at the input of the pulse generator 618 and at one input of this XOR gate 620 the error signal 38 is brought (in the same way as for the generation of the *main_clock 37*).

Comparing the architecture of Fig. 6 with the asynchronous one, the asynchronous speculative acknowledge left signal output 121 is now connected to the request left signal input 123, and the non-speculative request right signal output 113 is now connected to the acknowledge right signal input 111. In the right interface, the non-speculative request signal 113 is connected to the acknowledge right signal input 111 (instead of using the speculative request) because the *correct* flip-flop 615 stores the non-speculative data output 616 and this operation must be completed before a new *main_clock* pulse 37 is generated.

Further implementation details will not be described, as the man skilled in the art is able to carry out the invention starting from the teaching of the above description.

Many changes, modifications, variations and other uses and applications of the subject invention will become apparent to those skilled in the art after considering the specification and the accompanying drawings which disclose preferred embodiments thereof. All such changes, modifications, variations and other uses and applications which do not depart from the spirit and scope of the invention are deemed to be covered by this invention.

For example the error correction part of the method and circuit of the invention can be considered as optional with respect to the part relating to the meta-stability detection and recovery.

CLAIMS

1. Method for solving meta-stability conditions in a digital integrated circuit, comprising the following steps:

- detecting meta-stability conditions in the digital integrated circuit;
- using the information on said meta-stability conditions to temporarily stop the clock generation for the digital integrated circuit;
- restarting the clock generation for the digital integrated circuit when meta-stability conditions have been resolved.

2. Method as in claim 1, comprising a further step of signal error recovering, comprising:

- detecting error conditions in the digital integrated circuit;
- repeating the computation performed by the digital integrated circuit until the error conditions have been resolved.

3. Method as in claim 2, wherein said digital integrated circuit comprises a number of interconnected asynchronous blocks, said method comprising the steps of:

- exchanging the error condition information between said interconnected asynchronous blocks;
- temporarily stopping and restarting the clock generation of an asynchronous block of said number of interconnected asynchronous blocks, based also on error condition information received from one or more upstream blocks of said number of interconnected asynchronous blocks.

4. Method as in claim 3, wherein said step of exchanging error condition information between said interconnected asynchronous blocks in based on an handshake procedure.

5. Method as in claim 3 or 4, comprising exchanging, between upstream and downstream interconnected stages, acknowledgment and request signals to communicate that the downstream stage is ready to receive a new data, and the upstream stage outputs a new data, said data being of a speculative type if possibly affected by errors, or of a non-speculative type, if not affected by errors.

6. Circuit for solving meta-stability conditions in a digital integrated circuit, said digital integrated circuit comprising at least a storage block (3), a clock generator (1) for the at least a storage block, a combinatorial logic (2) supplying the data to be stored in the at least a storage block, said circuit for solving meta-stability conditions

comprises:

- at least a meta-stability detector (33), configured to detect meta-stability conditions at the output of one or more storage elements (31) in said at least a storage block (3), and to supply meta-stability condition signals (34) to the clock generator;
- at least a circuit for stopping and restarting the clock generation of said clock generator, configured to receive said meta-stability (34) signals, to temporarily stop the generation of the clock signal when said meta-stability are indicative of the presence of a meta-stability, to restart the generation of the clock signal by the clock generator when said meta-stability conditions signals are indicative of the resolution of the meta-stability conditions.

7. Circuit as in claim 6, also comprising:

- at least an error condition detector (32), configured to detect error conditions between input and output levels of said one or more storage elements (31), and to supply error condition signals (38) to the clock generator;
- at least a circuit for causing the clock generator to repeat the generation of the clock events, allowing the re-computation of the erroneous data by the digital integrated circuit, when said error condition signals are indicative of the presence of the error conditions, and until the error conditions have been resolved.

8. Circuit as in claim 6, wherein said at least a circuit for stopping and restarting the clock generation of said clock generator comprises delay circuits (136, 137, 138, 170, 171, 172) configured to receive said meta-stability and error conditions signals, and to delay the generation of said clock signal for time periods necessary for solving said meta-stability conditions and recovering signal errors.

9. Circuit for solving meta-stability conditions and recovering signal errors as in claim 8, wherein said digital integrated circuit comprises a number of interconnected asynchronous blocks, said circuit comprising:

- input and output interfaces (11, 12), configured to exchange error conditions signals with one or more upstream or downstream blocks of said number of interconnected asynchronous blocks;
- further delay circuits (131, 141, 142) configured to cooperate with said delay circuits (136, 137, 138, 170, 171, 172) and with said input and output interfaces for temporarily stopping and restarting the clock generation of an asynchronous block of said number of interconnected asynchronous blocks based also on error condition information

exchanged with one or more upstream or downstream blocks of said number of interconnected asynchronous blocks via said input and output interfaces (11, 12).

10. Digital integrated circuit comprising a circuit for solving meta-stability conditions and recovering signal errors as in any of claims 6 to 9.

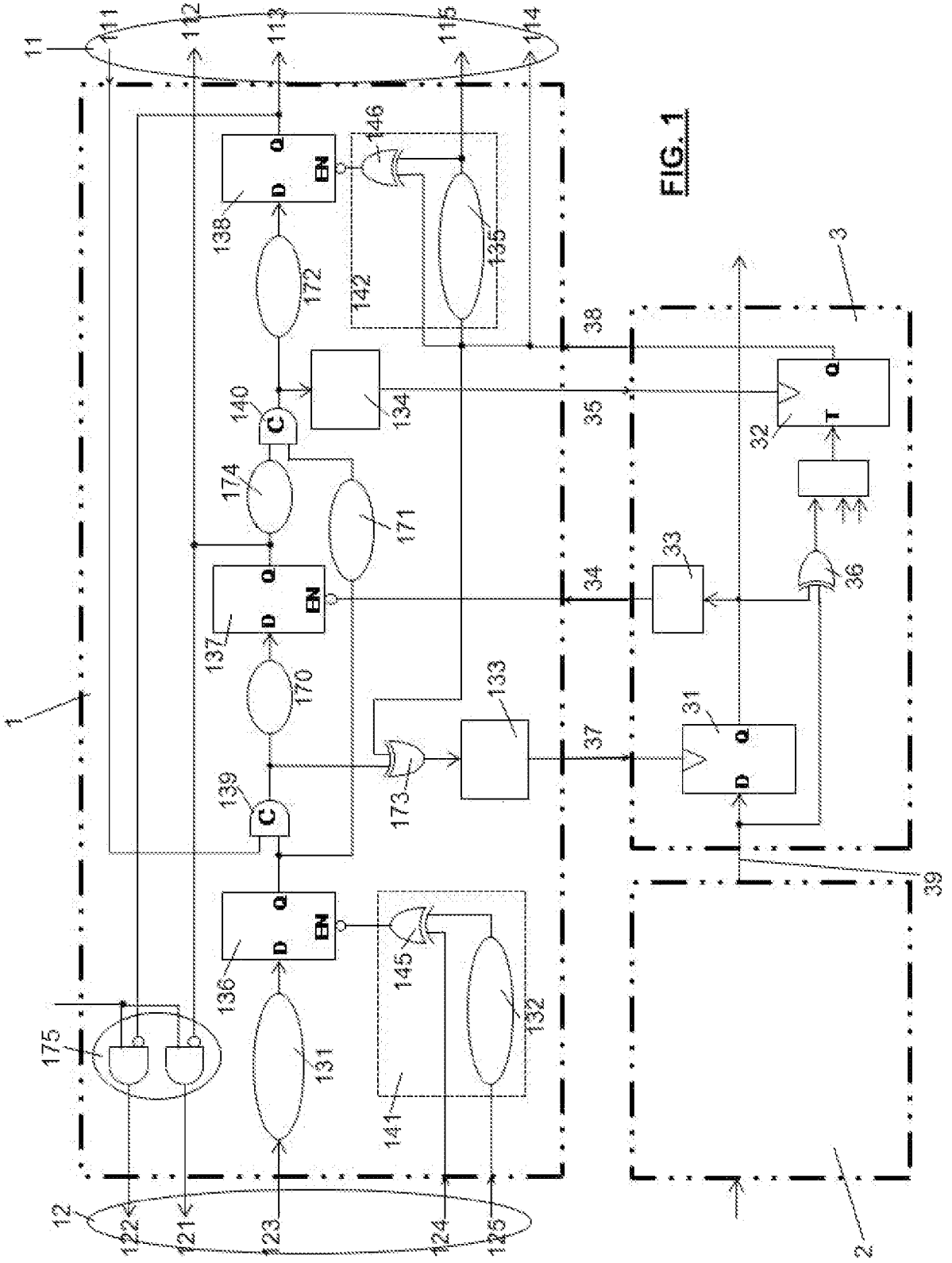
11
111
112
113
115
114
146
138
135
142
172
38
140
134
35
32
3
174
171
33
34
36
137
170
133
31
37
139
173
39
145
136
132
175
131
141
12
122
121
123
124
125
2
1

FIG. 1

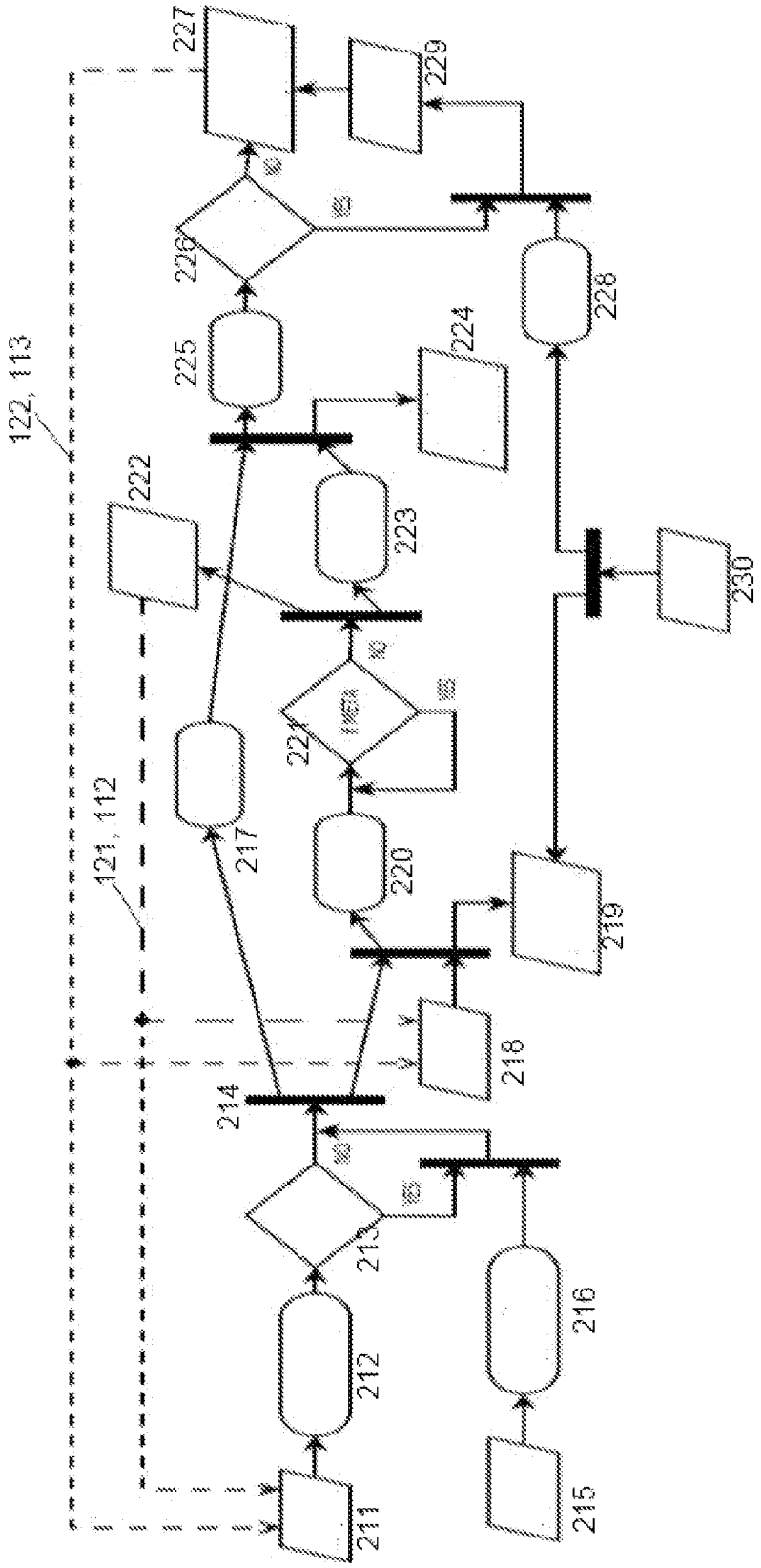
122, 113
121, 112
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230

FIG. 2

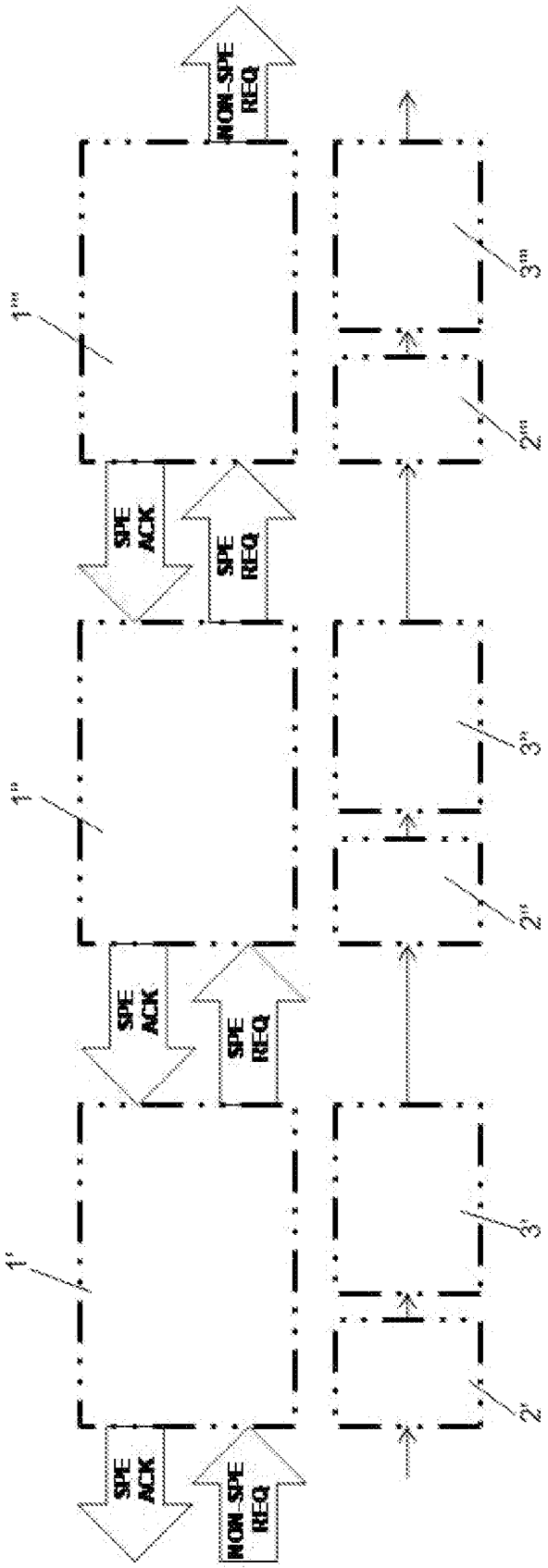

FIG. 3

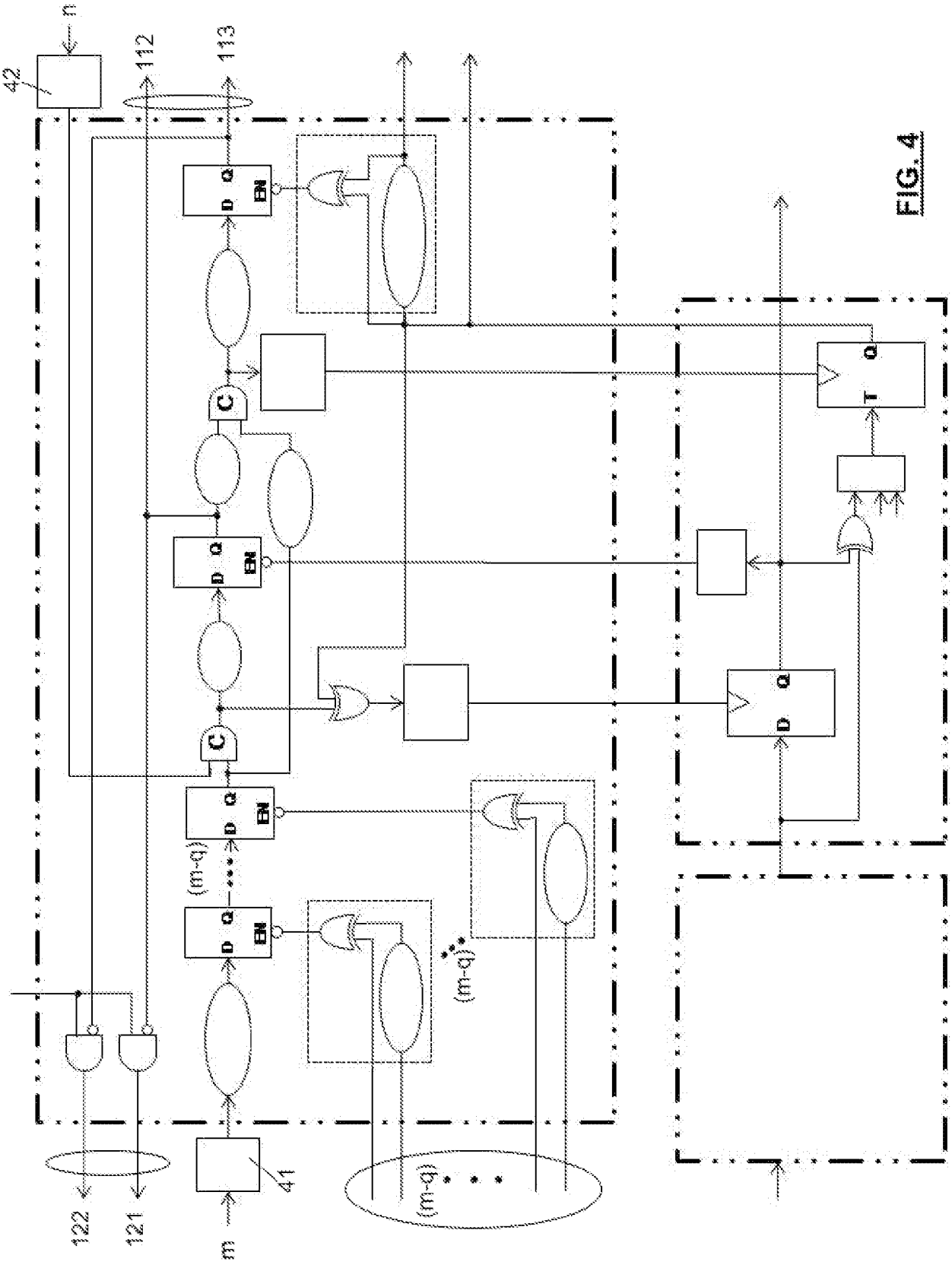

FIG. 4

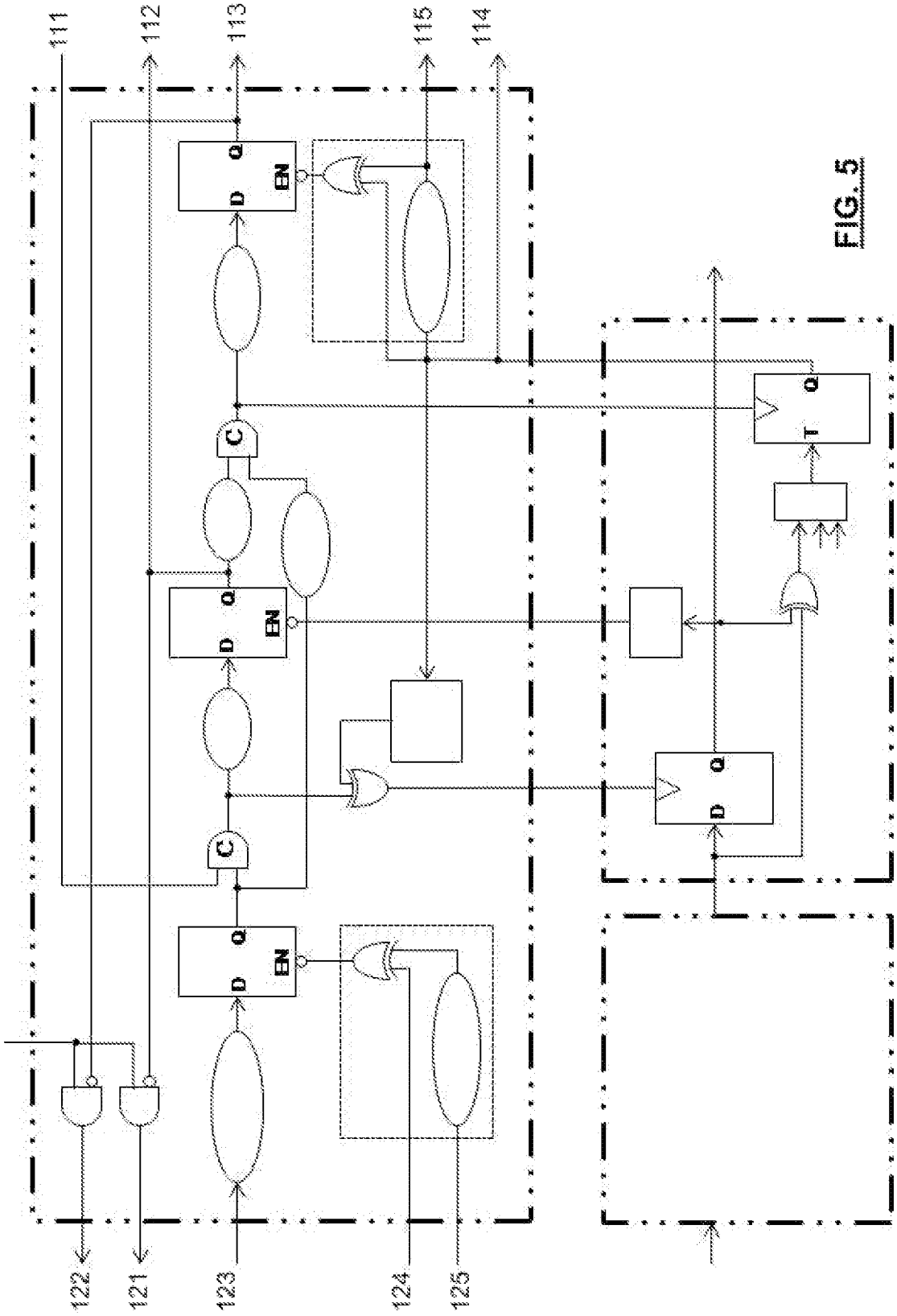

FIG. 5

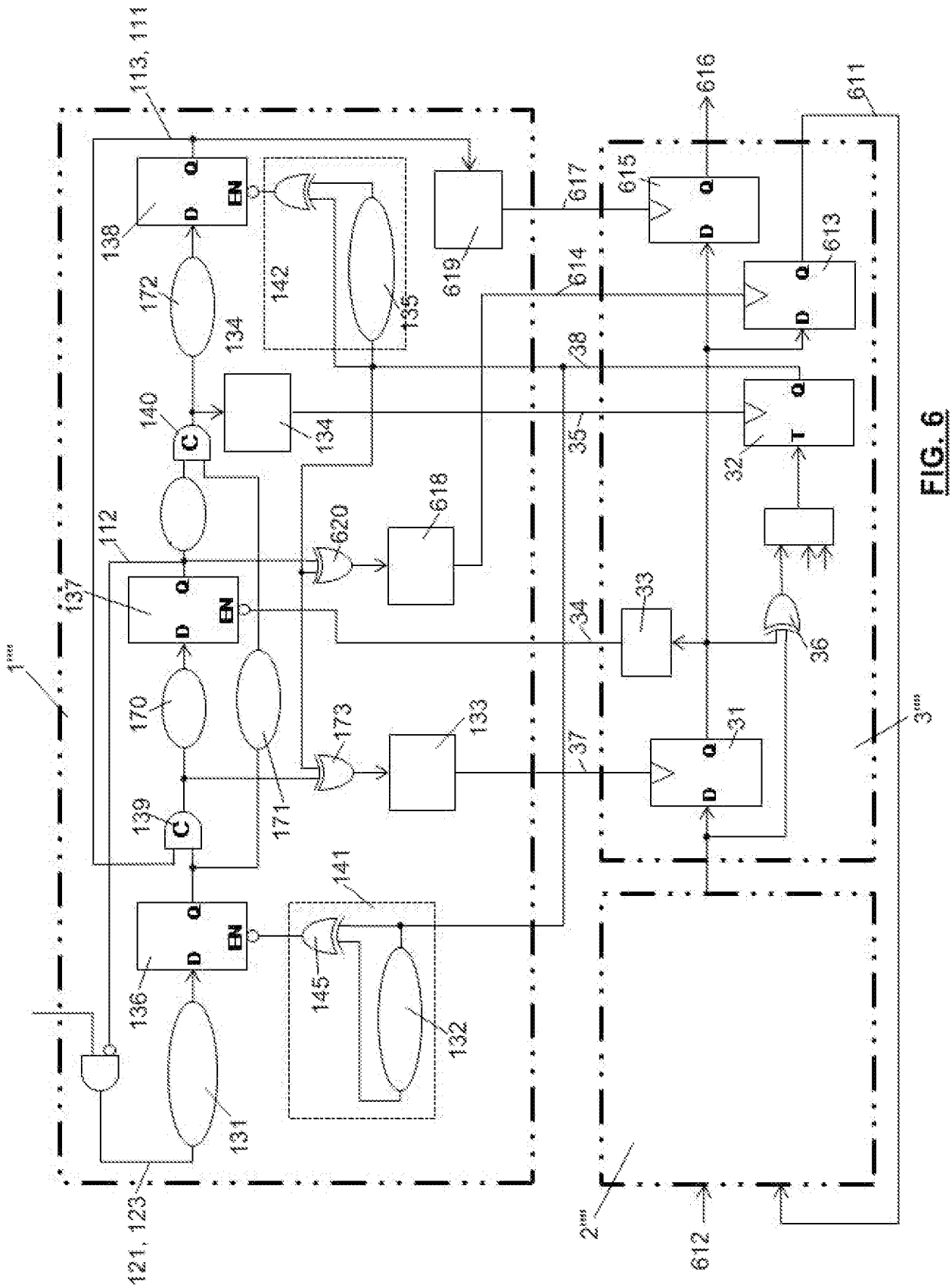
113, 111
138
172
142
135
134
619
616
611
617
615
613
614
38
140
134
35
32
618
620
112
137
33
34
36
1""
170
173
133
171
139
31
37
3""
141
145
136
132
131
121, 123
2""
612

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2012/052783

A. CLASSIFICATION OF SUBJECT MATTER
INV. H03K3/037 G06F11/00
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)
H03K 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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	LIM W Y-P ET AL: "Clocks and the performance of synchronisers", IEE PROCEEDINGS E. COMPUTERS & DIGITAL TECHNIQUES, INSTITUTION OF ELECTRICAL ENGINEERS. STEVENAGE, GB, vol. 130, no. 2, 1 March 1983 (1983-03-01) , pages 57-64, XP009156776, ISSN: 0143-7062, DOI: 10.1049/IP-E:19830014	1,6,10
Y	figure 3	2,3,7-9

Y	WO 2004/084070 A1 (ADVANCED RISC MACH LTD [GB]; UNIV MICHIGAN [US]; AUSTIN TODD MICHAEL []) 30 September 2004 (2004-09-30) cited in the application figures 2,3,19,20	2,3,7-9

	-/--	

[X] Further documents are listed in the continuation of Box C.

[X] 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
4 September 2012	14/09/2012

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	Santos, Paulo

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2012/052783

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2009/150706 A1 (OH MYEONG-HOON [KR] ET AL) 11 June 2009 (2009-06-11) figure 2	1-10
A	US 2009/024888 A1 (KURIMOTO MASANORI [JP]) 22 January 2009 (2009-01-22) the whole document	1-10

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/IB2012/052783

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2004084070 A1	30-09-2004	DE 602004001869 T2	03-05-2007
		EP 1604281 A1	14-12-2005
		JP 4317212 B2	19-08-2009
		JP 2006520954 A	14-09-2006
		KR 20050117559 A	14-12-2005
		US 2005022094 A1	27-01-2005
		WO 2004084070 A1	30-09-2004
US 2009150706 A1	11-06-2009	KR 20090061515 A	16-06-2009
		US 2009150706 A1	11-06-2009
US 2009024888 A1	22-01-2009	US 2009024888 A1	22-01-2009
		US 2011029829 A1	03-02-2011
		US 2011138217 A1	09-06-2011
		US 2011296260 A1	01-12-2011