Achieving degradation tolerance in a hardware accelerator with parallel functional units

Tomohiro Yoneda
National Institute of Informatics

Masashi Imai
University of Tokyo

Hiroshi Saito
University of Aizu

Atsushi Matsumoto
Tohoku University

Abstract

Recent advances in semiconductor process technologies cause new types of faults, which should be carefully considered for developing large and dependable VLSI systems. This paper focuses on a type of fault that degrades performance of circuit components. Conventional synchronous circuits need large margins to tolerate it. Our approach is based on asynchronous circuit technology with detection and data flow control mechanism to send less data to degraded units. The proposed idea is implemented in a linear equation solver. The simulation results show that the proposed method effectively tolerates the degradation of several functional units.

Key Words: *Degradation, NBTI, asynchronous circuits, linear equation solver*

1 Introduction

As semiconductor process technology scales and device dimensions shrink, new types of faults become a key reliability concern. One of such new faults is performance degradation of circuit components, which can be caused by effects like PMOS transistor negative bias temperature instability (NBTI)[1, 2, 3], hot carrier degradation (HCI)[4], V_{DD} drop, temperature increase, and so on. In order to tolerate such a fault, the conventional synchronous circuits need to have enough design margin, but it gives undesirable influence on system performance.

This paper focuses on those degradation faults, and proposes a mechanism to tolerate them without severe performance penalty. A well-known way for it may be the dynamic voltage and frequency scaling (DVFS) technique for synchronous circuits. In this approach, the degraded area or core is detected, and then, the supply voltage for the core is increased, or its clock frequency is decreased. The drawback of this approach is that the additional circuitry for degradation detection and the change of the supply voltage and/or the clock frequency is complicated, and the control for them is not simple. Instead, our approach proposed here is based on asynchronous circuit technology where no global clocks are used. Since an asynchronous circuit detects the completion of the computation in a functional unit, and the results of the functional unit are latched in the output register using the completion signal, the results latched are always correct, even if some degradation fault occurs in the functional unit. Thus, timing margins that are needed for synchronous circuits are no longer necessary.

It, however, does not mean that the degradation fault tolerance is achieved by asynchronous circuit technology alone. That is, the results produced from the degraded unit are correct, but it takes more time to produce them, which causes the completion of the whole computation to be delayed. For avoiding it, some mechanism to detect degraded parts and send less data to them is necessary. Such mechanism can be considered in several design levels. In the most coarse design level, asynchronous multi-core system can perform tasks or threads depending on their loads under the control of an operating system. On the other hand, a hardware accelerator with several functional units may be in one of the fine-grained design levels. In this level, care should be taken in order to minimize the overhead for the detection and the data flow control. In this paper, a mechanism for the latter case is discussed.

2 Targeted Architecture

In our approach, some of the processing power in a degraded unit should be compensated by normal functional units. Thus, our framework assumes that the original architecture has several or many units with the same functionality. One example is shown in Fig.1. In order to show the proposed idea as concretely as possible, one example is chosen and used as a running example throughout this paper. The proposed idea is, however, not limited to this particular example. For instance, it is easily applied to SIMD(Single Instruction Multiple Data) type architecture.

Suppose that $\boldsymbol{Ax} = \boldsymbol{b}$ is a linear system of equation, where $\boldsymbol{x}= (x_0, x_1, \cdots, x_{n-1})^T$ is a vector of n variables, $\boldsymbol{b}= (b_0, b_1, \cdots, b_{n-1})^T$ is a constant vector, and $\boldsymbol{A}$ is n-by-n coefficient matrix with non-zero diagonals. Linear equation solving is to compute $\boldsymbol{x}$ for given $\boldsymbol{A}$ and $\boldsymbol{b}$. If $\boldsymbol{A}$ is diagonally dominant, that is, $|a_{ii}| > \sum_{j \neq i} a_{ij}$ holds for every i, then the Gauss-Seidel method is one of simple and efficient methods for the hardware implementation of the

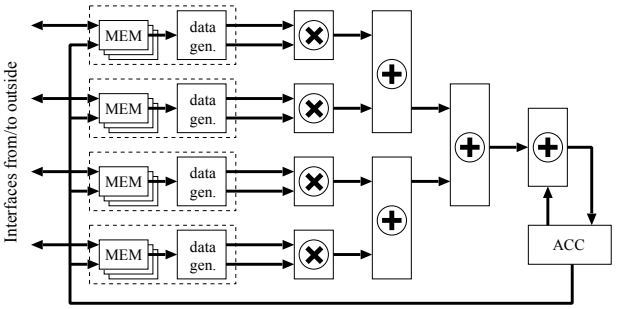

Figure 1. Example of a targeted architecture.

linear equation solver. It computes $x_i^{(k+1)}$, the value of x_i in the $(k+1)$-th iteration, as follows.

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^k)$$

The iteration stops, if the difference between x_i^k and $x_i^{(k+1)}$ becomes smaller than some threshold value for every i.

If each of n/m columns of $\boldsymbol{A'} = (A'_0, A'_1, \cdots, A'_{n-1})^T$ with

$$A'_i = (\frac{-a_{i0}}{a_{ii}}, \cdots, \frac{-a_{i(i-1)}}{a_{ii}}, \frac{1}{a_{ii}}, \frac{-a_{i(i+1)}}{a_{ii}}, \cdots \frac{-a_{i(n-1)}}{a_{ii}})$$

is stored in a different memory unit with copies of $\boldsymbol{x}$ and $\boldsymbol{b}$, and m multipliers are prepared as shown in Fig. 1 ($m = 4$ in this figure), then the above computation for $x_i^{(k+1)}$ can be done in an m-multiplex manner with respect to j. This is our example.

Our idea for degradation tolerance is to use only these multiplex functional units that are prepared for the performance improvement. No redundancy for improving reliability is necessary. This is the difference from a conventional approach using spare units.

3 Asynchronous Circuits

In asynchronous circuits, instead of using global clocks, request and acknowledgment signals perform local handshaking. Our work uses both bundled data and dual-rail encoding methods for sending and receiving data.

In the bundled data method (Fig.2), a request signal is used to show that the data is valid in the sender side. When it is asserted, the receiver starts to use the data. When the data processing is completed, an acknowledgment signal is asserted by the receiver, which allows the sender to destroy the current data and put the next data. For asserting the request and acknowledgment, our work uses so-called two-phase signaling, where both the rising and falling edges are used as shown in the above figure. In this bundled data method, a delay element is used to decide the timing when the acknowledgment signal is asserted, which should be matched

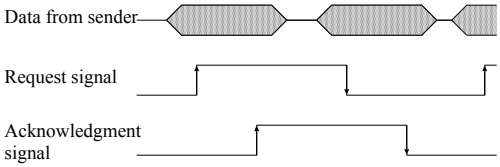

Figure 2. Bundled data method.

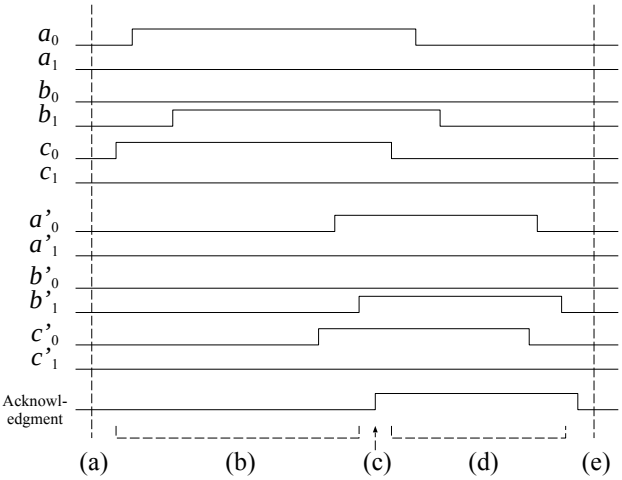

Figure 3. Dual-rail encoding method.

to the data transfer and processing time. The data-path circuits themselves are the same as those used in synchronous circuits.

In the dual-rail encoding method, a request signal is not used explicitly. Instead, the data is encoded by a dual-rail code, which allows the receiver to detect the completion of the data transfer and processing. The dual-rail code uses two signals for representing one data bit, i.e., (01) and (10) represent the data bit "0" and "1", respectively. (00), which is called *spacer*, is used for separating a sequence of data. This method works as follows. Initially, every data-path signal is reset to 0 (Fig. 3 (a)). When the sender sends a data, each data-path signal changes to 0 or 1 gradually. Similarly, the receiver gradually sees the change from (00) to either (01) or (10) for each data bit (Fig. 3 (b)). Note that this change should be monotonic, i.e., the change such as (00) $\rightarrow$ (10) $\rightarrow$ (01) should not happen. This is an important requirement for the data-path circuit used for this method, because it is needed to make the completion detection possible. The circuit implementation with monotonic property is not difficult. When the receiver detects that the signal pair for every data bit has either (01) or (10), it raises an acknowledgment signal (Fig. 3 (c)). Here, a level signaling (or 4 phase signaling) is used for the acknowledgment signal. Then, the sender resets every data-path signal to 0. This change is again propagated to the receiver gradually (Fig. 3 (d)), and when it is detected that the signal pair for every data bit has (00), the acknowledgment signal is lowered by

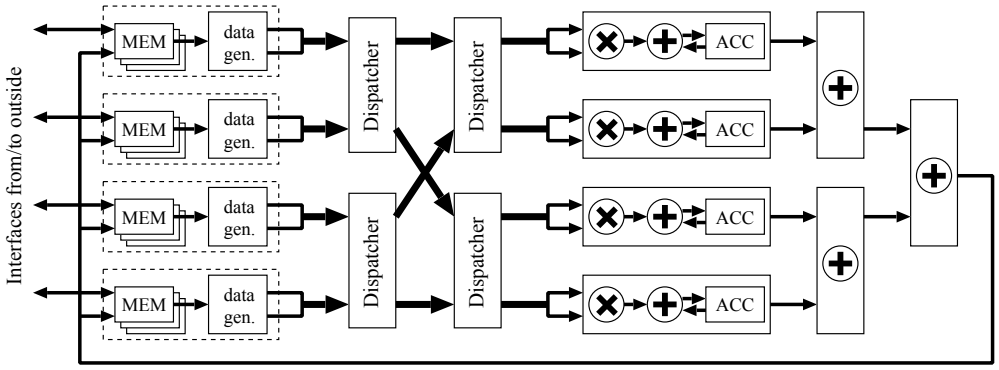

Figure 4. Proposed architecture.

the receiver, where every signal is in the initial state and the next data processing is ready (Fig. 3 (e)).

The dual-rail encoding method requires a special data-path circuit which is 1.5 to 2 times larger than the one used for synchronous design. However, it can exactly detect the completion of data processing. Thus, even in the case that the data-path circuit is degraded, when the results are produced, they are detected and latched correctly. On the other hand, in the bundled data method, delay elements are used to indicate the completion of data processing. It is unlike that both the data-path circuits and the delay elements suffer from the degradation similarly. Thus, if only a data-path circuit is degraded and the output of the result is delayed, an incorrect data may be latched.

In this work, the dual-rail encoding method is used for functional units that perform complicated data manipulation such as addition or multiplication, where the influence of the degradation is considered to be large. For the portions that handle data transfer without manipulation, such as dispatchers (see Fig. 4) and memories, the bundled data method is used, which means that the degradation of those portions is not assumed.

4 Proposed Method

4.1 Basic Idea

Thanks to the dual-rail encoding method, even degraded functional units can always produce correct results. However, in the architecture shown in Fig.1, the degradation in one functional unit can cause the delay of the completion of the whole computation. This is because each data is assigned to particular functional units in the architecture, i.e., the degraded functional unit is used as often as the other normal (non-degraded) units. Hence, we first releases the connection between data to be processed and the functional units.

Fig.4 shows the proposed architecture obtained by applying the above idea to the one shown in Fig.1. The dispatcher receives data from either of two directions, and sends it to either direction. Since the adder tree shown in Fig.1 always needs data from the both directions, the influence of the degraded functional unit cannot be separated. Thus, the adder is included in each functional unit in the proposed architecture, and the adder tree is used once when the final results are obtained.

The requirements to achieve the degradation tolerance of the functional units in this architecture are as follows.

1. The dispatcher should send less data to the direction where the degraded functional unit is located. However, sending data only to the normal functional units may not be appropriate. If the degraded unit can perform some computation, it should be used. The data load balancing is important in order to reduce the delay of the whole computation.

2. Functional units need to know the end of data in order to send the accumulated data to the adder tree. This is because the number of data to be processed is not fixed due to the load balancing.

4.2 Dispatcher design

The data-path circuit of the dispatcher is shown in Fig.5. It has two input ports and two output ports. A data that arrives at one input port is latched by either one of the two resisters that are tied to the output ports, and the latched data is sent out from the corresponding output port. Note that the bundled data method is used here. Thus, when data is sent out, the request signal from the selected output port is asserted, and the data is kept valid until the corresponding acknowledgment signal is asserted by the receiver. We say that an output port is *busy*, when the request is asserted but the acknowledgment is not yet asserted. A busy input port is defined similarly.

In order to satisfy the above requirement 1, the idea used in our dispatcher is very simple; it is to send the received data to one of non-busy output ports. This achieves the above requirement 1 naturally. Note that it is so simple thanks to the local handshaking of asynchronous circuits.

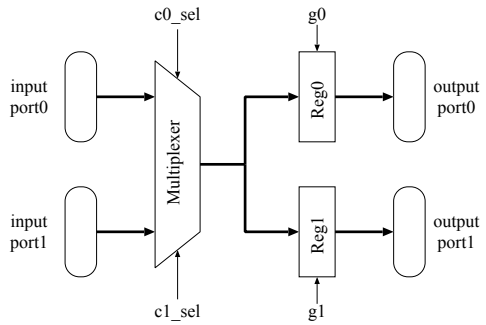

Figure 5. Data-path of dispatcher.

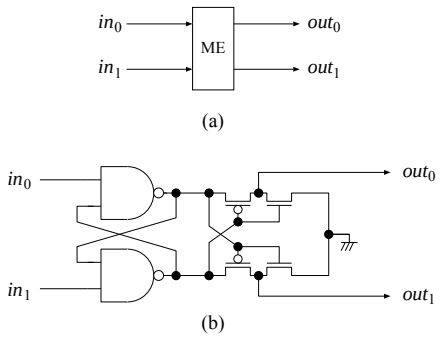

Figure 6. Mutual exclusion element.

If both output ports are busy at the time point when a new data arrives, the new data cannot be latched by any register. Thus, the input data should be kept by the sender. This is also simply done by not asserting the acknowledgment signal to the sender. Latching this new data and asserting the acknowledgment signal to the sender happen immediately when either output port becomes non-busy.

It is possible that two output ports become non-busy almost at the same time, or both are non-busy when an input data arrives. In order to choose one asynchronously, a mutual exclusion (ME) element is needed. An ME element is needed also for the input data selection. That is, when both input ports receive data, one of the data is selected for handling, and the other waits until the former data is latched. The ME element (Fig.6 (a)) has two input signals and two output signals. If in_0 (in_1) becomes high, out_0 (out_1) becomes high. If both in_0 and in_1 become high almost at the same time, either out_0 or out_1 becomes high. Both outputs never go high in the ME element. It is usually implemented using a RS latch with a metastability absorber (Fig.6(b)).

From the above discussion, the dispatcher should behave as follows.

1. It waits until either input port becomes busy, and either output port becomes non-busy.
2. One busy input port is selected by an input ME element, and one non-busy output port is selected by an output ME element.
3. The data-multiplexer shown in Fig.5 is controlled by the result of the input ME element.
4. The request signal for the selected non-busy output port is asserted. This output port becomes busy, and the data through the data-multiplexer is locked in the register for the output port. This also releases the output ME element.
5. The acknowledgment signal of the selected input port is asserted. This input port becomes non-busy, and the input ME element is released.

A little detailed (but not final) design of the control circuit of the dispatcher is shown in Fig.7. $in0\_req$ and $in1\_req$ are the request signals for the input ports, and $in0\_ack$ and $in1\_ack$ are their acknowledgment signals. Similarly, the output ports have the request signals ($out0\_req$ and $out1\_req$) and acknowledgment signals ($out0\_ack$ and $out1\_ack$). In order to detect the transitions of those signals, exclusive ORs and exclusive NORs are used, i.e., if $c0$ is high, then the input port0 is busy, and if $g0$ is high, then the output port0 is non-busy. Thus, those signals are connected to an input ME element and an output ME element, respectively. The outputs of those ME elements are connected to an OR-AND circuit to implement the above behavior 1 and 2. The delay element connected to the OR-AND circuit is for delaying the latch signal (c_clk) until the data through the multiplexer becomes valid. Although c_clk is given to the FFs for the both output ports, only one of them with $g0\_sel = 1$ or $g1\_sel = 1$ actually toggles. Note that toggling the request signal means its assertion in the two phase signaling. This makes the selected output port busy, and the data is locked in the corresponding output register (shown in Fig.5), which is a transparent latch, by lowering $g0$ or $g1$. The c_clk is also used to assert the acknowledgment signal of the selected input port by giving a rising edge in $tog0$ or $tog1$ to either toggle FF.

4.3 Detecting end of data

In order to inform the functional units that data are no longer sent to them, we use a *delimiter* that is a special data indicating the end of data. Since raw data (instead of data packets) are used in our design, the MSB of data bits is used for distinguishing delimiters from normal data.

When a functional unit receives a delimiter, it simply needs to send the accumulated data to the adder tree. The behavior required for the dispatchers is, however, a little bit complicated. Even if a delimiter arrives at one input port, the other input port may receive more normal data, which should be handled in a normal way. Thus, the dispatcher should wait until another delimiter arrives at the other input port. If both the input ports receive delimiters, then the dispatcher should send delimiters through both output ports, in order to indicate that no more data are sent from this dispatcher. Thus, the dispatcher should behave as follows for handling delimiters.

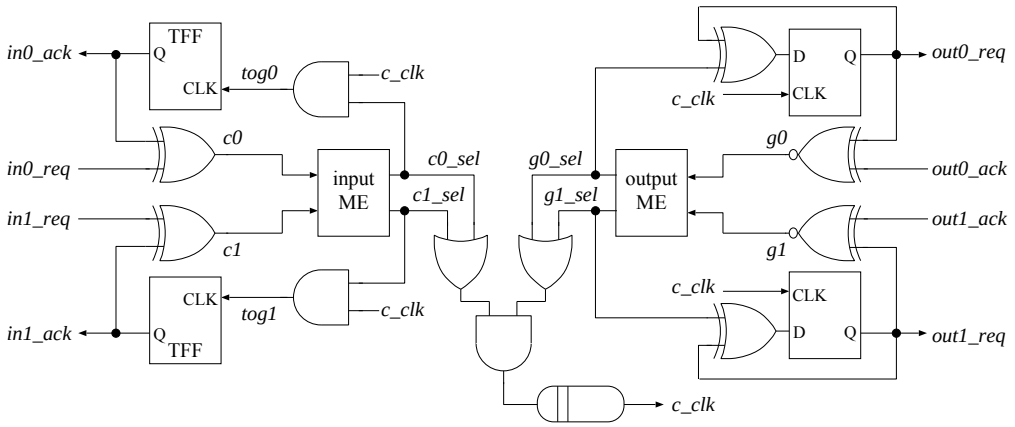

Figure 7. Dispatcher controller design.

1. When a delimiter arrives at one input port, its fact is recorded, and the acknowledgment signal for the input port is asserted.
2. When both input ports receive delimiters, the dispatcher waits until both output ports become non-busy.
3. Then, delimiters are sent out through both output ports.

The control circuit shown in Fig.7 is modified to handle the above behavior. The details are, however, omitted here.

4.4 Functional Units

A functional unit uses the dual-rail encoding method, while the outside of the functional unit uses the bundled data method. Thus, conversion circuits are needed in both the borders. Fig.8 shows the data-path circuit of our functional unit with its conversion circuits. This functional unit includes a multiplier and an adder that are connected in a pipeline style.

The single-rail encoded data sent from the input side is first converted to the dual-rail encoded data by combining the normal data bit and the inverted data bit to form a dual-rail encoded signal pair. A pair of a normal AND gate and an AND gate with an inverted input is used for each data bit to generate the dual-encoded signal pair as shown in the left dashed box of Fig.8. $spacer\_cnt$ signal of the converter is used to give the spacer to each signal pair. A dual-rail arithmetic circuit, such as a dual-rail multiplier and a dual-rail adder, is implemented using dual-rail AND, OR, NOT gates, which guarantee the monotonicity of the circuit. The completion detector produces a $compl$ signal by a circuit shown in the right dashed box, where it goes high when all signal pair is either (01) or (10), and goes low when all signal pair is (00). Finally, the single-rail encoded data for the result is obtained simply by taking the normal data bit from the dual-encoded signal pair.

Fig.8 also shows the control signals of the functional unit with the circuits producing some of those control signals. The control circuits use the idea of MOUSETRAP pipeline style[5]. The controller of the multiplier part behaves as follows. Note that transparent latches are used for registers.

1. When in_req is asserted, the controller waits until the multiplier becomes idle (i.e., $EN\_in = 1$). Then, it asserts in_ack through a transparent latch.
2. This makes EN_in go down, and the received data is locked in the register. At the same time, $spacer\_cnt1$ is raised in order to start the computation.
3. When $compl1$ goes high, meaning the multiplication completes, mul_req is asserted by giving a pulse to mul_tog.
4. mul_ack is asserted, and the lowered EN_mul locks the multiplication result in the input register for the ACC. It means that the multiplication result can be destroyed. Thus, $spacer\_cnt1$ is lowered to initialize the multiplier.
5. When $compl1$ becomes low, every dual-rail signal has (00) now. Thus, a pulse is given to in_tog, and EN_in goes high.

The controller of the ACC part is similar except that its request signal (not shown in Fig.8) is asserted only when a delimiter is detected. For normal data, the adder result is just stored into the ACC output register when $compl2$ goes up, and a pulse is given to acc_tog when $compl2$ goes down, in order to request the next data from the multiplier.

5 Experimental Results

A linear equation solver with 32 16-bit-variables and $m = 4$ is implemented using the proposed idea. A $0.13um$ process technology is used for the design, and the evaluation is done using the simulation results for the place-and-routed design. In order to simulate the degradation of functional

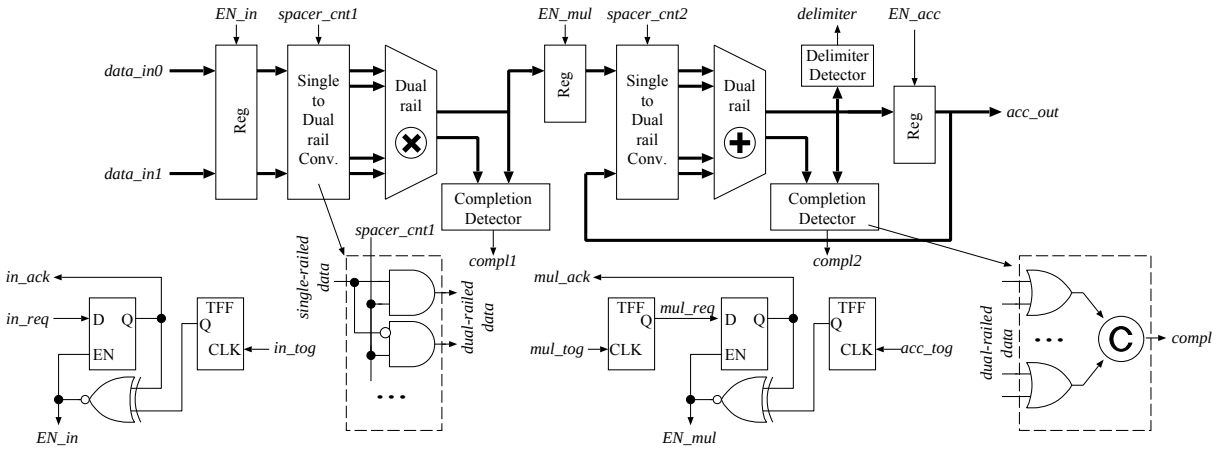

Figure 8. Data-path circuit for of a functional unit.

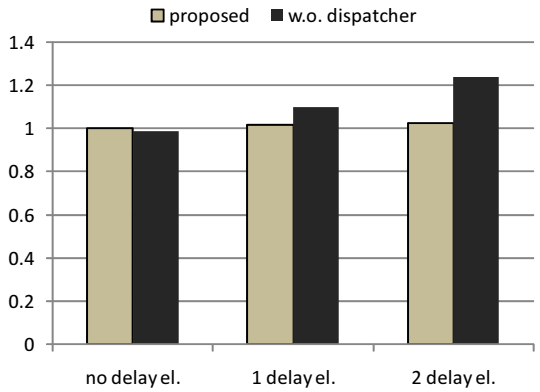

Figure 9. Experimental results.

units, delay elements are inserted in the acknowledgment signals of the functional units. Shortcuts to directly connect the data generators to the functional units are also prepared, which are used for estimating the performance without dispatchers. Those delay elements and shortcuts are activated through the control signals given from the outside.

Fig.9 shows the normalized performance where the performance of the proposed design without any delay elements is 1. It shows the performance degradation of the proposed design and the design without dispatchers (by shortcut), when delay elements are inserted to one or two functional units. As shown in the figure, the proposed method has high ability to tolerate degradation.

6 Conclusion

This paper proposes a method to give degradation tolerance to a hardware accelerator with parallel functional units. It is based on asynchronous circuit technology and data flow mechanism controlled by the acknowledgment signals to the output ports. A linear equation solver is implemented using the proposed idea, and according to the simulation results for the place-and-routed design, the proposed method shows much better degradation tolerance than a simple asynchronous implementation. Our future work includes the application of the proposed method to other and larger design examples.

Acknowledgment

This work is supported by CREST (Core Research for Evolutional Science and Technology) of JST(Japan Science and Technology Agency). This work is also supported partially by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc and Cadence Design Systems, Inc.

References

[1] J. W. McPherson. Reliability trends with advanced CMOS scaling and the implications for design. *Proc. CICC2007*, pages 405–412, 2007.

[2] S. Basu et al. Process variation and NBTI tolerant standard cells to improve parametric yield and lifetime of ICs. *Proc. ISVLSI2007*, pages 291–298, 2007.

[3] K. Kang et al. Characterization of NBTI induced temporal performance degradation in nano-scale SRAM array using Iddq. *Proc. ITC2007*, pages 1–10, 2007.

[4] T. H. Ning. Hot-carrier emmission currents in n-channel IGFET's. *Int. Election device Meet. Tech, Dig.*, pages 144–147, 1977.

[5] Montek Singh and Steven M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. In *Proc. International Conf. Computer Design (ICCD)*, pages 9–17, November 2001.