Better Than Worst-Case Computing

Why not to miss the typical case

Oliver Klöckner

Sommerakademie in Leysin, August 2016

Abstract

Nowadays it is common for fast and modern computer systems to work on a GHz level. But since the size of transistors reached about a few nanometer, the microchips and processors have gotten smaller and smaller. Because of their size, such are more likly to be influenced by environmental conditions. The Worst Case design overcomes those impacts at the expense of energy consumption. But in contrary, going beyond this with Better Than Worst Case designs could make processors more energy efficient. Therefore severals techniques were invented to face this challenge and to reach higher clock rates or lower the voltage supply with just a little overhead. Allowing an error rate higher than 0 besides the other implementations, leads as well to a significant increase of speed. But still all of these methods are not suitabel for every system and have to face many challenges.

Contents

1 Introduction

One of the most important parts in a computer is the clock. It synchronizes all parts to a frequency f, all calculations and operations through the whole processing unit. Because of this, the frequency is an important factor for energy consumption P. But the main key is the supply voltage V, which influences the consumption even by square:

$$P = C * V^2 * f \tag{1}$$

The capacitance of the transistors C makes up the third factor.[2] So an energy efficient system should lower the voltage by keeping the same frequenzy for fast but efficient calculations.
But as microchips and microprocessors are getting smaller and already reached the low nanometer size, environmental influcences gain more importance because of their impact on the logic gates. For instance, with higher temperature the resistance of the electronic elements increases. In addition, often it is not possible to have the same production conditions at every manufacturing place. Therefore every microchip will behave in a different way. But with clock changes every a few nanoseconds, even a small delay could corrupt the data of a calculation. This happens because the gates do not have enough time to switch their state. Thus the clock is reliant on these influences. This is the reason, why ingenieurs of such systems have to include a specific delay or to increase the voltage to cover unintendend effects. And this is called Worst Case Design.[8] For Instance a Ripple Carry adder like in figure 6 has to wait for the result of the last full adder and this depends on the carry in of the previous ones. So the clock needs to wait for n full adder.
The developer of a computer systems adapt their products to certain fixed conditions to make sure they will work properly. But often they slow down the system, because they are adapted to a specific range of temperature, altitude or voltage supply. This, however, allows the system to work at worst case without any errors.
Despite the negative effects on the energy consumption there are plenty of reasons to use the Worst Case Design. One point is the very small size of the transistors, the very high clockrate and the huge impact of any environmental influences. But in some specific fields a correct working system is needed indeed. E.g. in financial sector, in public transportation or in aerospace, where systems are more likely to reach their limits. Besides this, setting fixed conditions allows the developer of microchips to be safe, which also speeds up developement and makes it cheaper, especially when the field of usage is unknown like for general purpose. And still there is the believe that if it works in worst case fine, it will work even better in the avarage, but vice versa it could end catastrophically.[5]
First of all I want to give an introduction into the field of the Worst Case design. In the second part I will refer to several techniques, which are used to avoid the worst case. In the end there will be a summary along with the challenges this design has to face.

2 Better Than Worst Case

There are several approaches to overcome the worst case design. Most of them follow a common pattern called the Better Than Worst Case Design. And in this section 3 of them will be introduced: the Razor logic, the Typical Case optimization and a rather different one with approximation.

2.1 Better Than Worst Case Design

The Better Than Worst Case design covers various techniques to get beyond the worst case specification without any errors at all. Figure 1 presents such a design. The aim is to allow the core component to work outside the worst case specification without any impacts. Therefore the Checker is seperated from the optimized core component. Because the checking is done seperatly, it is less complex to be implemented. But there are also special demands for it: the Varified Checker needs to be easy to be implemented, has to be able to check even under worst case conditions and should be free of any errors. Has it hit all those requirements, it is possible to adapt the frequency and supply voltage to the current error rate. So all in all, if the frequency will be increased or the voltage will be decreased, the checker will be able to detect an error and adapts to one of them.[7]

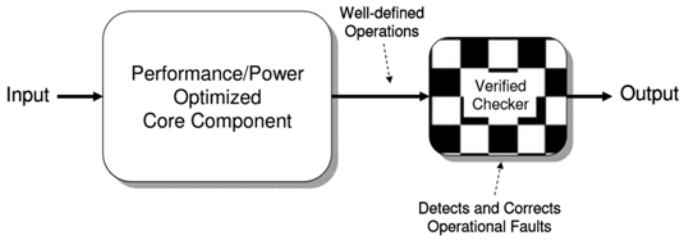

Figure 1: Better Than Worst Case Design Concept[7]

2.2 Dynamic Voltage Scaling - Razor Logic

Decreasing the supply voltage is the most efficient way to lower the energy consumption. Implementing this, however, can lead into wrong calculation results because of delays. Therefore you have to check wether the result you get are correct or not. The Razor Logic implements this by appending appending a control logic right after the main logic like in figure 2. The logic consists of a main flip-flop and a shadow latch, a comperator and a switch. The output of the main flip-flop is connected to the next stage of calculation and to a XOR gate as well as the output of the shadow latch. The XOR gate controlls the switch, which toggles between the logic input and the output of the shadow latch.
The main flip-flop is controlled by the main clock, which goes faster than under the worst case. The shadow latch, however, depends on a shadow clock, that is delayed to the main one to manage the worst case. The main flip-flip saves the input signal only on a rising clock. When the result from the logic will be recieved in time, the main flip-flop will store it on the next rising clock. Likewise, the shadow latch will store the same result during a delayed high clock signal. Because of this, the input signals of the XOR gate are equal and no error occured. But when the calculation was too slow due to a low voltage supply, the results could differ. Eventhough the main flip flop could store a wrong result on rising clock, the shadow latch will look after this. In that case, the input signals of the XOR gate can differ, so an error occurs. As this happens, the state from the shadow latch will be passed to the input of the main flip flop for the next clock.[7][3][6]

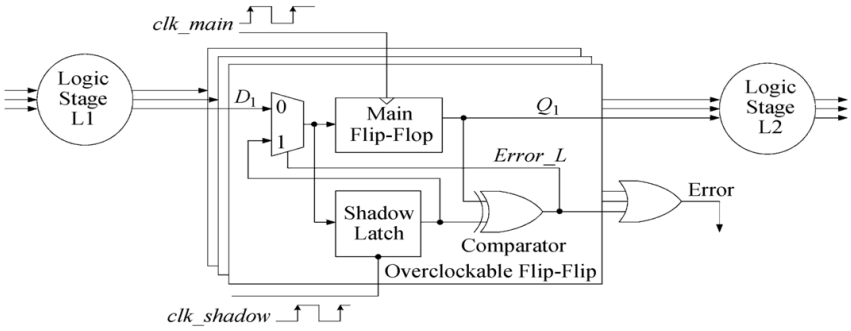

Figure 2: Razor Logic[3]

Besides the logic, the error signal signal will also be passed to the pipeline stages. The reason for this is the avoidance of corrupted data in all the stages. To back this up, there are two different concepts[6]:

1. Clock Gating, like in figure 3, stalls the memory storage process after an error signal for one clock cycle. So the correct state can be passed from the shadow latch to the main flip flop. Then the pipeline continues normaly.

2. Counterflow, like in figure 4, avoids stalling the pipeline for one clock. This will be used, when waiting is not possible due to an aggrissve pipeline design. The error signal, instead, is causing a bubble signal, which will be flushed thorugh the pipeline stages without any processing. After this the execution stage will be processed again.

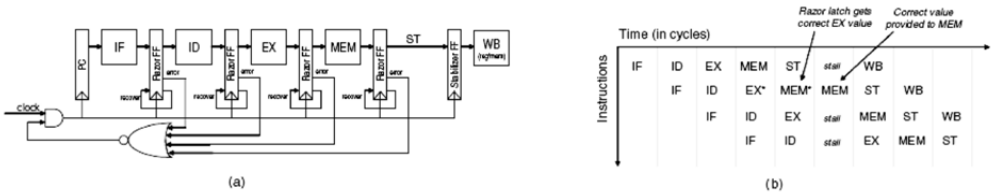

Figure 3: Clock Gating[1]

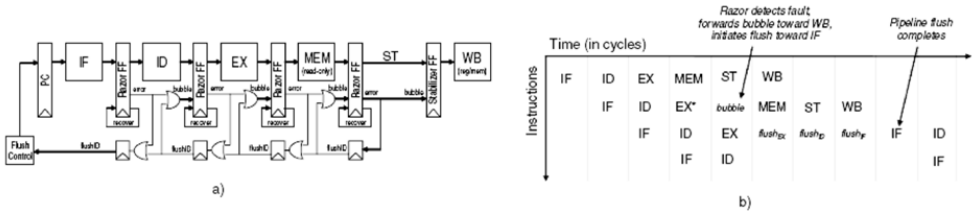

Figure 4: Counterflow[1]

Finally, to adapt the voltage dynamically, you need a controlling system, that adapts the current voltage to the error signal. Such a controlling process is outlined in figure 5. The error signals from the processing step are summed up to a sample error rate, which is compared to a reference one, e.g. 0%. Based on this, a voltage control function regulates the voltage for the pipeline.

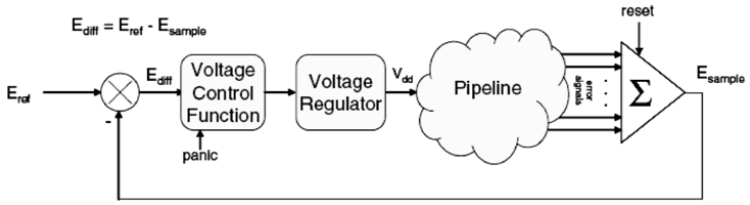

Figure 5: Dynamic voltage scaling process[1]

Mainly, the Razor is logic combined with dynamic voltage regulation to compensate environmental influencens as well as production ones. Apart from that, it can be used to go beyond the worst case specification for the voltage supply in runtime. In 2004 this system was tested on modified ARM cores with the SPEC2000 benchmark. The logic reached great energy savings up to 64% by only producing 3% overhead for error correction.[1]

2.3 Typical Case Optimization

Another important approach, to face the worst case design, is called Typical Case Optimization. In opposite to the Worst Case design, the TCO tries to adapt the logic to the typical case with a greater penalty in the worst case scenario. In fact the Razor logic is already a TCO solution, because less used parts of the electronic circuits will get a lower voltage supply at runtime. There are 3 steps to implement the TCO[7]:

1. Evaluation of the duration under typical conditions, e.g. with example data
2. Comparison to the general/random case
3. Optimization of the circuits

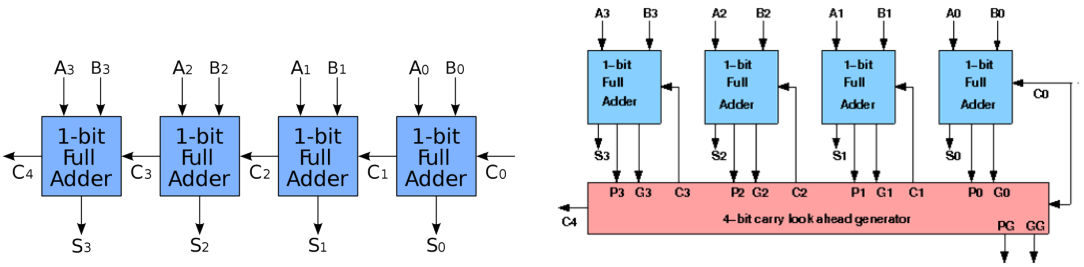

Figure 6: Ripple Carry Adder link and Carry Look Ahead Adder link

To illustrate this process deeper in detail, a Ripple Carry adder will be analysed and implemented in a Carry Look Ahead one[7], like in figure 6. A Ripple Carry adder consist of n 1 bit full adder, which are connected through the carry in/out. Even though the n single bits can be added in parallel, the whole adder has to wait for n carry in/out to ensure the correct result, shown in figure 6. But indeed, the Most Significant Bits have a higher impact on the results than the least ones. This is the reason why a TCO analysis should be done on the carry propagation. If the distances are far in the typical case, a special logic, e.g. a Carry Look Ahead adder, can be added. It will prefer them, vice versa for short distances. Therefore, the propagations for every bit position were evaluated within the SPEC2000 benchmark for typical sets of Add, Branch, Load and Store instructions. The same analysis was made with completely random data again to compare both and they can be found in figure 7 .

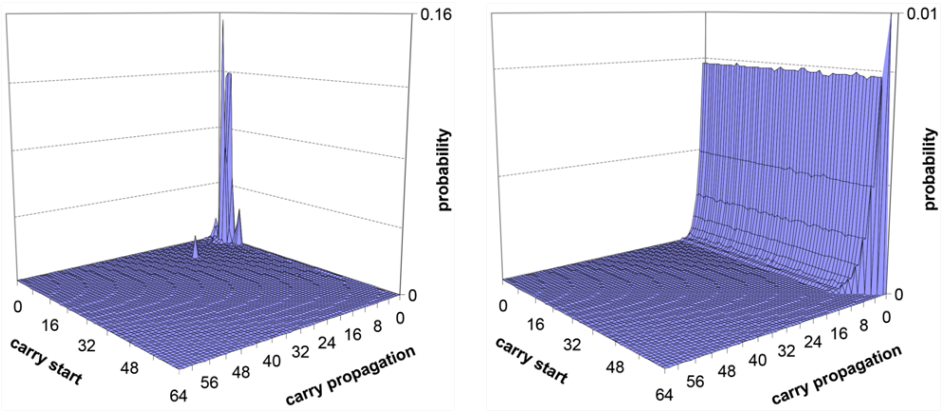

Figure 7: Typical set (left) and random set (right)[7]

Adder	**Latency** (in gate delays)		
Topology	Worst-Case	Typical-Case	Random
Kogge-Stone	8	5.08	7.09
TCO Adder	128	3.03	3.69

Figure 8: Comparison of TCO and Kogge Stone adder[7]

As a result, the typical data set has mostly short carry propagation distances: in the least significant bits less then 6 and in the most significant bits more than 2. Whereas in the random set of data the propagation is independent from the bit position. As a consequence from these result, an optimized version of the adder was build as a Carry Look Ahead and compared to an Kogg-Stone adder, which is already optimized for the worst case. Table 8 shows the comparison between both of them. The optimized version has a very high latency compared to the Kogg-Stone adder in the worst case, but in the typical and even in the random case the TCO implementation is rather faster than the Kogg-Stone, which is a huge performance gain.[7]

2.4 Approximate Computing Designs

The last method that will be presented is the Approximate Computing Design. In contrast to the previous ones, this concept allows an error rate above 0%. Therefore this approach is common used in AI or video applications where a lack of presicion is tolerable without any bigger impact on the results. There are 3 different ways to implement this design:[3]
Error Distribution Manipulation tries to reduce the energy consumption by reaching a specific error rate while lowering the voltage supply. A Ripple Carry Adder has a moderate error rate once the critical voltage supply was hit. Therefore this one good, if a moderate error rate is tolerated. Apart from that, the Kogge Stone Adder has a lower critical voltage than the other one, but after reaching it, the error rate grows rapidly. So it should not be used for any error rate above 0 at all. Because of these behaviours, the right method should be used at runtime considering the desired error rate. By implementing this Method in the Kernel of the H.264 video codec, energy savings from 20% up to 60%.[3]
The second method, Probabilistic Pruning, focuses on reducing less likely active parts of the electrical circuits. To get those, first of all the probability for the usage of any of them will be calculated based on simulations oder mathematical models. Then those circuits will be reduced and the error

rate will be calculated. If the rate is to high, the reducing process will be made undone, otherwise it will be continued.
The last method uses a configurable approximative adder to adapt the error rate to the quality and speed during runtime. Because the MSB has the biggest influence on the result at addition, guessing the MSB based on the few previous ones uses less carry out information. The MSB is also less likely to be influenced by more distant bits. Therefore a higher throughput can be reached. For instance, with this method JPEG coding could reach a 21% higher throughput.[3]

3 Challenges

The Worst Case design is indeed a limiting factor for higher clock rates and less energy consumption. But otherwise, error free systems for critical sections would not exist. Better Than Worst Case designs, however, show that it is possible to go beyond those restrictions. Having the hardware implementations on the one hand side, we can still have an error free system. On the other hand side, allowing approximation on non critical sections achieves significant savings without any complex hardware implementation at all. But still it is hard to implement approximation in general purpose processors. Through both of these processes, adapting to runtime is a crucial thing. Besides this, Worst Case design does not mean your system cannot get faster without the BTWC design. In fact it is possible to gain a boost without any big adjustmens. E.g. if you are using your computer at home, you will often have similar and stable conditions, that are not beyond the worst case. Therefore it is possible to overclock some CPU and GPU without any problems. With special cooling it is also possible to go even further than 8 GHz.[1]

References

[1] S. Das S. Pant R. Rao Toan Pham Ziesler C. Blaauw D. Austin T. Flautner K. Mudge T D. Ernst, N. S. Kim. Razor: a low-power pipeline based on circuit-level timing speculation. *Proc. 36th Annual IEEE/ACM Int. Symp. on Microarchitecture (MICRO-36)*, 2003.

[2] Intel. Why p scales as c*v^2*f is so obvious: link. 2009.

[3] Rakesh Kumar Sen Li Jason Cong, Henry Duwe. Better-than-worst-case design: Progress and opportunities. *Journal of Computer Science and Technology*, 10.1007/s11390-014-1457-2, 2014.

[4] Nils Meyer, Manfred Ries, Stefan Solbrig, and Tilo Wettig. iDataCool: HPC with hot-water cooling and energy reuse. *CoRR*, abs/1309.4887, 2013.

[5] Rui Zhang-Shen Nandita Dukkipati, Yashar Ganjali. Typical versus worst case design in networking. *Computer Systems Laboratory, Stanford University.*

[6] Margaret Martonosi Stefanos Kaxiras. Computer architecture techniquise for power-efficiency. *Synthesis Lectures on Computer Architecture*, 2008.

[7] David Blaauw Todd Austin, Valeria Bertacco and Trevor Mudge. Opportunities and challenges for better than worst-case design. *IEEE*, 10.1145/1120725.1120878, 2005.

[1] Overclocking Ranking

[8] August K. Uht. Going beyond worst-case specs with teatime. *IEEE Computer Science*, 10.1109/MC.2004.1274004, 2004.