Towards Scalable Arithmetic Units with Graceful Degradation

DANNY P. RIEMENS, Netherlands Institute of Neuroscience and Delft University of Technology
GEORGI N. GAYDADJIEV, Chalmers University of Technology and Delft University of Technology
CHRIS I. DE ZEEUW, Erasmus Medical Center and Netherlands Institute of Neuroscience
CHRISTOS STRYDIS, Erasmus Medical Center and Delft University of Technology

This article presents a new family of scalable arithmetic units (ScAUs) targeting resource-constrained, embedded devices. We, first, study the performance, power, area and scalability properties of general adders. Next, suitable error-detection schemes for low-power embedded systems are discussed. As a result, our ScAUs are enhanced with a suitable error-detection scheme, resulting in a Parity-Checked ScAU (PCScAU) design. The PCScAU strikes a flexible trade-off between space and time redundancy, offering dependability similar to high-end techniques for the area and power cost of low-end approaches. An alternative design, the Precision-Scalable Arithmetic Unit (PScAU) maintains throughput with degraded precision in case of hardware failures. The PScAU is targeting dependable applications where latency rather than numerical accuracy is more important. The PScAU's downscaled mode is also interesting for runtime thermal management due to its advantageous power consumption. We implemented and synthesized the PCScAU, PScAU and a few important reference designs (double-, triple- and quadruple-modular-redundancy adders with/without input gating) in 90-*nm* UMC technology. Overall, the PC-ScAU ranks first in 9 out of 10 power-delay-area (PDA)-product variants. It exhibits 16% area savings and 12% performance speedup for 7% increase in total power consumption, compared to the cheapest form of conventional hardware replication with the same fault coverage. The PDA product of the PCScAU is, thus, reduced by 21%. It is interesting that, while total power slightly increases, the PCScAU static power in fact decreases by 14%. Therefore, for newer technology nodes where the static power component is significant, the PCScAU can also achieve—next to performance and area – significant power improvements.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Fault-Tolerance; Reliability, Availability, and Serviceability

General Terms: Design, Experimentation, Performance, Reliability

Additional Key Words and Phrases: Computer arithmetic, scalable design, graceful degradation, fault tolerance, error detection, error correction, low power consumption, embedded systems

ACM Reference Format:
Danny P. Riemens, Georgi N. Gaydadjiev, Chris I. de Zeeuw, and Christos Strydis. 2014. Towards scalable arithmetic units with graceful degradation. ACM Trans. Embedd. Comput. Syst. 13, 4, Article 87 (February 2014), 26 pages.
DOI: http://dx.doi.org/10.1145/2499367

The work has been partially supported by: the EU-funded project DeSyRe (Grant agreement no: 287611), the Dutch Organization for Medical Sciences (ZonMw) and Life Sciences (ALW), Senter (Neuro-Basic) and the ERC of the European Community.
Author's addresses: D. P. Riemens, Netherlands Institute of Neuroscience, Meibergdreef 47, 1105 BA, Amsterdam, Netherlands; G. N. Gaydadjiev, Chalmers University of Technology, Dept. of Computer Science and Engineering, Rännvägen 6, S-412 96 Gothenburg, Sweden; C. Strydis and C. I. de Zeeuw, Dept. of Neuroscience, Erasmus Medical Center, Dr. Molewaterplein 50, 3015 GE, Rotterdam, Netherlands; email: c.strydis@eramusmc.nl.

DOI: http://dx.doi.org/10.1145/2499367

1. INTRODUCTION

With the advent of mobile computing largely enabled by the rapid technology-miniaturization trends, modern embedded applications place demands for ever lower power budgets on computer architectures [Mudge 2001]. Low energy has also become a first-order design constraint, particularly for battery-operated devices. However, the same shrinking device features that have—at first—brought individual transistor power consumption down, have also led to increasing power density per chip unit area, larger process variations [ITRS 2011] and accelerated aging [Hazucha et al. 2003], to name a few of the problems. The result of such effects in the coming nanoscale era is chips with diminishing reliability due to the growing number of hard and soft faults. Maintaining correct functionality in the presence of high fault rates is bound to increase manufacturing costs and drive chip lifetimes down [Borkar 2005].

While these phenomena are manifesting in the mainstream market, research and industrial niches have been struggling to build low-power embedded systems with high fault tolerance for many years now. These niches commonly include mission-critical systems for military, space, biomedical and other applications. In such systems, the effects of the shrinking technology trends pose serious dangers for designers who respond by staying with older and safer technology nodes [Strydis 2011].

In line with these observations, the primary objective of this work is to propose a new family of gracefully degradable (through scalability) and low-power, integer arithmetic units (AUs) for embedded systems. The reason we focus on AUs is that they are one of the busiest components in low-power, embedded [Segars 1997] and high-performance [Shrivastava et al. 2010] processors alike. They are, thus, a source of high power consumption/dissipation as well as high fault sensitivity. On the other hand, the properties of many AU designs permit striking efficient trade-offs between power consumption, performance and fault tolerance. Such characteristics are their inherent scalability and particular error patterns, discussed later.

In this work we, first, perform a comparison of various popular adder designs synthesized in a modern technology node (UMC 90 *nm*, Standard Purpose), followed by a comparison of error-detection/-correction (ED/EC) techniques suitable for low-power, scalable AUs. Based on the findings of the two concise surveys we propose new gracefully degradable AU designs with 100% single-fault and 50% double-fault coverage while ranking first among state-of-the-art competitors. Evaluation has been based on the empirical cost metric Power-Delay-Area product (PDA) and its variants. The PDA metric, used to characterize combinational circuits, factors in the power consumption and area cost of a design as well as its propagation delay.

While this work broadly addresses low-power, embedded processors, our current research is focused on biomedical, microelectronic implants (Smart implantable Medical Systems - SiMS [Strydis 2011]), which is an excellent case of severely resource-constrained embedded systems with high reliability requirements. The findings reported in this work are generic yet, without loss of generality, when needed we will explicitly borrow from implant specifications (e.g., AU width) to demonstrate the properties of our proposed adders. Thus, the following original contributions are made.

—We conduct a survey on the power, performance, area and scalability characteristics of various standard integer-AU types, implemented in modern CMOS technology.
—We conduct a survey on ED/EC techniques suitable for application in the AU of low-power embedded systems.
—We present a new family of low-power, low-area and fault-tolerant scalable AUs which outperforms state-of-the-art AUs.
—We present adaptations of this AU family which can trade arithmetic precision for performance as well as thermal dissipation.

The remainder of this article is organized as follows. In Section 2 related work is discussed. In Section 3, relevant background information is provided as well as a study among general adders implemented in modern technology. Then, in Section 4, the scalable-arithmetic-unit (ScAU) idea is presented: the design of the ScAU, the data path, and the synthesis results are explained. Also, the ScAU's synthesis results are compared with competitive fault-tolerant adder structures based on redundancy. A variant of the ScAU, the precision-scalable-arithmetic unit (PScAU) is, then, discussed. Finally, the effects on the ScAU design when using fast-adder building blocks are investigated and a number of low-power design techniques is proposed to lower the ScAU power consumption. In Section 5, several error-detection and -correction schemes, suitable for protecting adders, are discussed. A detailed comparison between residue (or modulo-3) and parity (or modulo-2) checking is presented. Two different implementations of the parity-checking scheme are, then, compared and the most suitable one is selected in the context of the targeted application domain. Section 6 presents the fault-tolerant ScAU. Finally, in Section 7 the concluding remarks of this work are drawn.

2. RELATED WORK

2.1. Scalable, Power-Proportional Arithmetic Structures

Scalable arithmetic units, ALUs and micro-architectures in general have been extensively studied. For example, Lee [2005] presented a scalable Booth-multiplier. The objective was to reduce the power consumption, by scaling down the multiplier width, when both input operands contain small values (detected by a dynamic-range detection unit). A similar design is presented by Pfänder et al. [2008] focusing on flexible word-length multiplication to save energy.

Kumar et al. [2005] presented a design for an energy-efficient, high-performance architecture (including functional units, register files, caches, etc.), where the data path is bit-sliced. The higher order bit-slices are only activated if required, which saves a significant amount of energy. Iyer and Marculescu [2001] also worked on a scalable micro-architecture. The resources are dynamically allocated, following the running-program needs. Here, each basic block of code (instruction sequence without jumps or branches) is analyzed determining the parallelism and resource usage and the processor is reconfigured for each basic block. Any unused FUs are disabled using clock gating. In Lin et al. [2005] a low-power ALU cluster design was presented, which is basically an ALU that is composed of multiple clusters (2 AUs, 2 multipliers and a divider), each placed onto a separate voltage island. That means, that any of the clusters can be power-gated when not in use, leading to higher power and energy savings than the regular clock-gating mechanism. Also, Monteiro et al. [1996] reported the so-called data-dependent power shut down, which is used in the Intel-Pentium family of processors. Even though these implementations do not have a flexible word length, they all can be considered as scalable designs in terms of power consumption.

Another form of power-scalable designs exclusively used in arithmetic units, are mechanisms to vary the arithmetic-operation precision. An example was given in Amirtharajah et al. [1999] using distributed arithmetic, but many different implementations exist. The idea is that, whenever possible, the arithmetic unit does not produce exact but, relaxed-precision results.

Besides, there are more motivations for scalable designs. Kursun et al. [2005] have described the problems occurring when on-chip temperatures increase excessively. In such cases the lifespan of the device is compromised due to the temperature-induced accelerated aging and operating speeds are slowed down. Dynamic-thermal-management techniques have been developed to cope with these problems by limiting the heat dissipation so as to avoid reaching critical chip temperatures. Since power consumption and heat dissipation are two sides of the same coin, scalable designs have been employed for keeping the latter in check as well.

2.2. Fault-Tolerant Adders

Fault tolerance means that apart from detecting (certain) errors, the circuit is also capable of correcting them. According to the literature, ED/EC methods to achieve fault tolerance (at the architectural level) can be facilitated through: (i) *hardware replication*, (ii) *time redundancy*, (iii) *ED/EC codes*, and (iv) various combinations of the above techniques.

The most widely used ED technique is hardware replication. It has been utilized for many years due to its high fault coverage. One can duplicate a circuit and feed both the circuit outputs to a comparator; in case of comparison mismatch, an error is detected. Such systems with resource replication are also called duplex or Duplication-With-Comparison (DWC) [Mitra and McCluskey 2000; Parhami 2000; Sellers et al. 1968].

An example of a fault-tolerant adder using hardware replication is the TMR-adder (Triple Modular Redundancy). Here, three adder units are present, operating simultaneously and feeding their results to a majority voter. This voter looks for a match between any two of the three inputs and outputs that result. Another fault-tolerant adder based on hardware replication is the QMR-adder (Quadruple Modular Redundancy) [Townsend et al. 2003]. Two adders and a comparator form together a Self-Checking Adder (SCA). Since, it is impossible to obtain information about which of the two adders inside a SCA is faulty in case of error, the QMR contains two SCAs, with both outputs attached to a multiplexer. In total, the QMR scheme contains four adders. The fault coverage of these fault-tolerant adders is very high. However, the power and area costs of the TMR and QMR are often excessive because of the replicated adder structure and extra checker circuitry needed.

Another type of fault-tolerant adder employing hardware redundancy is described in Khedhiri et al. [2012] and uses alternative computations. In every bit slice of the adder, the sum and carry are computed in two different ways and compared with each other. In case of an error, one of the redundant components is brought in to continue the calculations, saving 11.1% in transistor count compared to the QMR adder.

Peng and Manohar [2005] presented an asynchronous adder which achieves fault tolerance through dynamic self-reconfiguration. The adder is built based on a fault-tolerant array. In case of a fault, reconfiguration logic will try different configurations until a workable instance is found. Depending on the degree of fault coverage (single faults to quadruple faults), the area overhead of these self-healing adders varies between 102% and 326% and is claimed to be lower than traditional redundancy methods (TMR and QMR). Note that this technique can be also employed in sequential adders at the cost of extra ED logic.

Cardarilli et al. [2006] employ the properties of radix-2, signed-digit representation to build a fault-tolerant adder. Repair is based on graceful degradation: either by recomputing the result with shifted operands and utilizing the intersection of the obtained results to recover the correct output or based on a reduced-dynamic-range approach, where the result is obtained faster but with fewer output digits.

A low-cost, fault-tolerant technique for carry-lookahead adders was proposed by Namazi et al. [2009]. Correction of all single-bit and multiple-bit transient faults is claimed. The power and area costs of this method are significantly lower than those of the TMR adder or the duplicated adder with parity checking. The introduced additional delay is small, in particular compared to parity-checked adders. This technique is, however, only applicable for Carry-Lookahead Adders (CLAs).

3. BACKGROUND

In this section, a short survey of adders is performed studying performance, power, area and error characteristics. We implement all studied adders in a modern technology node and investigate how their design characteristics scale with different input widths.

3.1. Standard Adder Types

The most well-known and simplest adder is the ripple-carry adder (RCA). The RCA design is regular, easy to implement and with the lowest area and power costs among all existing adders. Its primary limitation is the long critical path, which grows linearly (O(n)) with the word width n. Faster adders with shorter delays but increased area and power consumption also exist [Parhami 2000]. The well-known types are:

—carry-lookahead adder – CLA; O($log(n)$);
—carry-select adder – CSA; O($\sqrt{n}$);
—carry-skip adder – CSK; O($\sqrt{n}$); and
—ripple-carry/carry-lookahead adder (hybrid adder) – RCLA; O($log(n)$).

For implementing a scalable-adder design, not only the delay, area and power consumption of the adder are important. The adder should also lend itself to *functionality segmentation*, that is, the ability to divide the adder up in segments, without compromising the functionality and benefits of the adder in question; it should, therefore, have a regular structure. Obviously, the RCA is always preferred in terms of power and area, when it is deemed fast enough for its intended purpose. It also has the most regular structure of all known adder types. The CLA, and in particular the RCLA hybrid, as well as the CSK deserve some further investigation, mainly because of their regular structures and their high-speed potential. The CSA will be omitted in our study due to two reasons. First, this adder has no regular structure which makes scalable implementation very difficult. Second, previous studies such as Rabaey and Pedram [1996] show that CSA overheads are considerably higher than those of the CLA while the delay increases.

3.2. Implementation of Standard Adders

Next, a study among different adder structures is presented. A number of motivations have led to this study. The first is to find the most suitable adder structure for implementation in the scalable arithmetic unit. Instead of duplicating adders, we implement a single adder, scalable in size. The basic idea we present in this article is to shut down only a part of the adder, namely that specific part where the failure has occurred, and continue the computational work using the remaining adder part(s). Not every adder structure is suitable for scalable-arithmetic-unit implementation, since the adder must be divisible into (at least) two segments, capable of operating independently and together as a whole (to be detailed in Section 4).

Second, a number of researchers have already compared different adders in the past such as [Callaway and Swartzlander 1992; Nagendra et al. 1996; Rabaey and Pedram 1996; Vratonjic et al. 2005]. However, in all studies the adders have been implemented in significantly older technologies, such as 2μm CMOS [Rabaey and Pedram 1996] and 1.2μm CMOS [Nagendra et al. 1996], so it was difficult to safely predict the delay and power consumption for much smaller, sub-micron technologies such as 90-*nm* CMOS. More importantly, we would like to verify whether the trends identified between the various adders in older technologies still hold in 90-*nm* CMOS. Consequently, we synthesized and compared various types of general adders to gain up-to-date insight on their delay, area and power characteristics when implemented in modern technology.

Table I. Results for 100MHz, random inputs, UMC 90 *nm* SP synthesis. Lowest values per input size and per characteristic are denoted in bold.

Component	Delay [ns]	Area [units]	Power [μW]
RCA 8-bit	0.79	**232**	**13.20**
CLA 8-bit	**0.52**	398	24.60
RCLA 8-bit	0.57	346	20.16
CSK 8-bit (4×2-bit blocks)	0.68	300	16.89
RCA 16-bit	1.48	**464**	**25.71**
CLA 16-bit	**0.68**	867	48.77
RCLA 16-bit	1.03	692	39.17
CSK 16-bit (8×2-bit blocks)	1.03	600	33.25
CSK 16-bit (4×4-bit blocks)	1.12	580	32.98
RCA 32-bit	2.86	**928**	**50.93**
CLA 32-bit	**0.90**	1734	97.92
RCLA 32-bit	1.95	1384	78.36
CSK 32-bit (8×4-bit blocks)	1.49	1312	74.02

Table II. Cost metrics (100 MHz, random inputs, UMC 90 *nm* SP technology). Lowest values per input size and per metric are denoted in bold.

Component	PA	PD	AD	PDA
RCA 8-bit	**3,062.40**	**10.43**	**183.28**	**2,419.30**
CLA 8-bit	9,790.80	12.79	206.96	5,091.22
RCLA 8-bit	6,975.36	11.49	197.22	3,975.96
CSK 8-bit (4×2-bit blocks)	5,067.00	11.49	204.00	3,445.56
RCA 16-bit	**11,929.44**	38.05	686.72	**17,655.57**
CLA 16-bit	42,283.59	**33.16**	**589.56**	28,752.84
RCLA 16-bit	27,105.64	40.35	712.76	27,918.81
CSK 16-bit (8×2-bit blocks)	19,950.00	34.25	618.00	20,548.50
RCA 32-bit	**47,263.04**	145.66	2,654.08	**135,172.29**
CLA 32-bit	169,793.28	**88.13**	**1,560.60**	152,813.95
RCLA 32-bit	108,450.25	152.80	2,698.80	211,477.97
CSK 32-bit (8×4-bit blocks)	97,114.24	110.29	1,954.88	144,700.22

3.3. Synthesis Results and Evaluation

The synthesis results for the various adder types are depicted in Table I, the cost metrics in Table II, and Table III depicts the speed/area/power increase with respect to the RCA. The cost metrics illustrated are typical figures of merit used when assessing digital circuits; for capturing all interdependencies between the various design aspects, we include here all first-order products of power, delay and area (PA, PD, AD, PDA). Since we are evaluating combinational circuits, the number of different input combinations is prohibitively large; we have, thus, used pseudo-random input vectors for our experiments. The input drive strength of the adder circuits has been set high, and the capacitive load on the circuit's output ports was set to 0.1 pF. We targeted UMC 90-*nm* Standard-Purpose technology.

We found that the RCA has the lowest PDA product, for all adder widths under investigation. Thus, despite the long delays, the RCA is the most cost-effective adder type. It can be seen from Table I that the CLA is the fastest adder and scores the lowest for both the Power-Delay product and the Area-Delay product for the 16- and 32-bit versions due to the very low delays. However, the power and area overhead of the CLA is very large. Table III shows the large differences in delay, area and power between the CLA, RCLA and CSK adders. Clearly, the CLA does not meet (ultra) low-power

Table III. Speed, area and power increase compared to RCA (100 MHz, random inputs, UMC 90 *nm* SP).

Component	Speed increase [%]	Area increase [%]	Power increase [%]
CLA 8-bit	51.9	71.6	86.4
RCLA 8-bit	38.6	49.1	52.7
CSK 8-bit (4×2-bit blocks)	16.2	29.3	28.0
CLA 16-bit	117.6	86.9	89.4
RCLA 16-bit	43.7	49.1	52.4
CSK 16-bit (8×2-bit blocks)	43.7	29.3	29.3
CLA 32-bit	217.8	86.9	92.3
RCLA 32-bit	46.7	49.1	53.9
CSK 32-bit (8×4-bit blocks)	91.9	41.4	45.3

and/or small-area demands. Also, the 16-bit CLA contains two levels of lookahead logic that is an irregular structure encumbering scalable design.

When comparing RCLA and CSK adders, we observe that the power and area increase of the CSK are—for all widths—significantly lower than those of the RCLA. The speed of the 8-bit RCLA is higher than that of the 8-bit CSK but for the 16- and 32-bit versions results are opposite. The conclusion can be drawn, then, that for 16- and 32-bit widths, the RCLA has no advantages over the CSK. Note that the 16-bit 4×4 CSK adder is less interesting than its 8×2 counterpart which is faster while requiring minimal extra area and power. Therefore, the 4×4 CSK is omitted in Tables II and III.

Finally, we also subjected the RCA, CLA and CSK to worst-case input sequences (in terms of power) while also considering *glitching power* in our study. Although absolute power numbers change, the relative ordering of the adders remains the same. Conclusively, there are no significant differences in trends between the older technologies used by Rabaey and Pedram [1996], Nagendra et al. [1996] and the more recent UMC $90 - nm$ Standard-Purpose technology.

3.4. Errors in Adders

Adders differ from all other logic structures because of the complications that occur when an error manifests itself in the carry circuitry. The basic element in adders is the full-adder. A full-adder is built by two (partially shared) circuits, to implement 'sum' and 'carry' [Parhami 2000; Sellers et al. 1968]:

$$s_n = a_n \oplus b_n \oplus c_{n-1} \tag{I}$$

$$c_n = a_n \cdot b_n + (a_n + b_n)c_{n-1}, \tag{II}$$

where (II) can be subdivided into a generate (III) and a transmit/propagate (IV):

$$g_n = a_n \cdot b_n \tag{III}$$

$$t_n = a_n + b_n, \tag{IV}$$

such that

$$c_n = g_n + t_n \cdot c_{n-1}. \tag{V}$$

A single fault in an adder circuit leads to an error that can fall into either one of two different categories: a single sum-digit error or a burst of sum- and carry-digit errors [Sellers et al. 1968]. A single error occurs when the fault is caused by a fault in a sum circuit, a burst error by a fault in a carry circuit. This makes the error characteristics of adders special, since burst errors normally only occur as a result of soft errors (discussed in more detail later).

A single error in a_n, b_n, or c_{n-1} will always cause an error in s_n (I). An error in a carry digit c_{n-1} will always propagate and cause an error in s_n as well. So, a fault in a carry circuit will cause two errors at a minimum: one carry-digit error and one sum-digit error. Now, if the situation is such that, due to the error, the incoming carry cannot be absorbed in stage n, carry digit c_n will be erroneous and so will be s_{n+1}. This way the burst error can become arbitrarily long, however, the total number of errors is always even. Because of this phenomenon, carry errors are of primary concern in adder ED [Sellers et al. 1968].

4. SCALABLE ARITHMETIC UNIT

In Section 2, several prominent scalable arithmetic structures and fault-tolerant adders have been presented. However, many existing solutions are not applicable to the domain of ultra low-power and highly resource-constrained embedded systems because of their high costs. In this article, we propose a new family of scalable arithmetic units based on a novel hybrid technique between space and time redundancy. In so doing, our proposal offers dependability similar to high-end techniques for the area/power cost close to low-end ones.

As discussed previously, a common technique for building reliable computing systems is hardware replication. For instance, if one adder fails, switching to a second spare adder maintains correct operation. Of course, this method has high cost in terms of power and area. Instead of duplicating the adder block inside the arithmetic unit, we aim at implementing a single adder block only, albeit now a version capable of changing the adder width to guarantee correct functionality. For example, an n-bit adder can be divided into two $n/2$-bit segments. When a single segment fails, the AU could proceed with the nonfaulty one. In such a case, a one-cycle, n-bit addition or subtraction will be replaced by two-cycle $n/2$-bit operations. The word size of the arithmetic unit is downscaled, even though the word size of the architecture remains the same. The adder could be divided into more than two segments which would further increase its reliability (the arithmetic unit would be able to proceed with the computational work, even when multiple adder segments are damaged). However, the more segments damaged the lower the remaining throughput will be. This scalable approach is an example of *graceful degradation*. We will henceforth call the *scalable arithmetic unit* ScAU.

4.1. ScAU Datapath

Primary concerns in the design of the ScAU are (i) an efficient *multiplexing network* to redirect the data (operands and results) via alternative paths, (ii) the *storage of intermediate results* for multicycle operations, and (iii) the *control logic* needed.

The data path of the proposed ScAU is depicted in Figure 1. The basis of the ScAU comprises two separate $n/2$-bit adders, A (left adder) and B (right adder). The lower $n/2$-bit part of the operands is fed to adder B, the upper $n/2$ bits to adder A. The carry-out signal of adder B is connected to the carry-in of adder A. Each adder has its own $n/2$-bit complementer and both the upper and lower half of the result are stored in a n-bit output register. This provides the basis for a single-cycle, n-bit operation. When one of the adders is faulty, the data must be rerouted. The adder, which is still intact, must be provided with the lower half of the operands in the first cycle, and with the upper

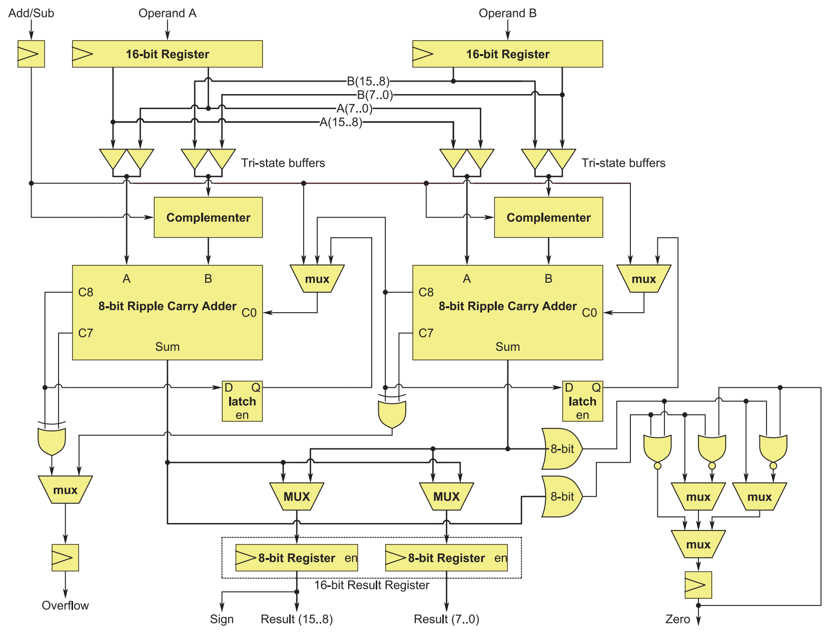

Fig. 1. The scalable arithmetic unit (2×8-bit instance) with overflow- and zero-detection circuits included. Tristates used for isolating any of the two 8-bit RCA blocks. Addition result placed in a 16-bit result register.

half of the operands in the second cycle (in downscaled mode). This is implemented by a multiplexer network. To keep track of the current cycle (cycle 1 or cycle 2), and to steer the multiplexer network, a controller is implemented. Also, a multiplexer network is required for the output of the adders. The output of the functioning adder should provide its result to the lower half of the result register in the first cycle, and to the upper half of the result register in the second cycle. Finally, additional latches are needed to store the value of the carry output of the functioning adder during the first cycle, to feed it to the carry input of the same adder in the second cycle. This means multiplexing at the carry inputs of both adders as well.

Furthermore, there is another important issue that requires attention. When the scalable AU is in downscaled mode, it will require two clock cycles per operation instead of one and, therefore, the amount of energy per instruction increases significantly. The most obvious way to limit the energy per instruction is to shut down the segment that is not used (prevent switching activity). For sequential circuits, *clock gating* is the easiest way to disable a (sub)circuit. However, adders are pure combinational circuits and, therefore, require another approach: *input gating*. This means keeping all of the inputs of the circuit constant to prevent any switching activity and thus limiting dynamic power consumption inside the circuit. *Power gating* would be another option for achieving this (with the advantage that also static power is limited), however in our specific case this is not beneficial, as will be explained later.

The initial design of the scalable AU is roughly subdivided into four parts: (i) the multiplexing and input-gating network, (ii) the adder logic, (iii) the output multiplexing logic, and (iv) the control logic. It is easy to see that the overhead of the ScAU mainly lies in parts (i), (iii), and (iv). In the initial design, guard latches [Kaxiras 2008; Tiwari et al. 1998] were used to disable unused combinational logic which, in this case, is any unused adder. It was, later, found that utilizing tri-state buffers for this purpose saves both power and area (at least, for UMC $90-nm$ SP technology). Also, a single tri-state buffer network can implement both the multiplexing and the input-gating functionality, which saves further area and power. Since no additional logic for input gating is required, input gating is preferred over power gating, as the latter would require extra logic. However, if even more stringent constraints are placed on power consumption when in downscaled mode and the static power component is considerably high (e.g., by using a more recent technology node), power gating might become interesting. It can be facilitated in the design quite easily since it requires only changes in the AU control logic.

Apart from the operand inputs, there is an input signal (1 bit) to select for addition or subtraction as well as a 'scale' signal (2 bits). This 2-bit signal is used to control the scalability feature of the AU (A and B adders functional, A-only, B-only, none functional). Finally, a special result register is required. This register must be able to load all bits in parallel, as well as load the lower half and upper half of the result individually. When the ScAU is in downscaled mode, the register should load the lower half of the result in cycle 1, store it, and load the upper half in cycle 2.

Note that the controller described above is not depicted in the datapath of Figure 1. Also, the ScAU is not fault tolerant yet. There must be error-detection and -correction logic present, in order to detect a failure and reconfigure the circuit. This topic is covered later, in Section 5. Without loss of generality, in this particular instance, the cheapest adder type is utilized for the adder blocks: the RCA.

4.2. Reference Designs

So far, the concept of the ScAU and its general implementation have been explained. In order to evaluate the ScAU, a 16-bit specific instance was implemented and compared with the following reference adder designs:

—a 16-bit AU with a single adder (no redundancy);
—a 16-bit AU with a duplicated adder (redundancy); and
—a 16-bit AU with a dupl. adder, where one adder can be disabled (by input gating).

A 16-bit width has been chosen for our evaluation purposes since it is a reasonable size for low-power, embedded systems [Hennessy and Patterson 2012, Appendix J.2]. It is also a nominal data size for most implant designs to date [Strydis 2011].

4.3. Synthesis Results and Evaluation

The synthesis results of the 16-bit ScAU, as well as the AUs with a single and a duplicated adder, are shown in Table IV. The acronym AU refers to the arithmetic unit implemented with a single adder, while DAU refers to the arithmetic unit implemented with a duplicated adder structure (where both adders are simultaneously active). The DAU-RAS is the arithmetic unit with a duplicated adder where the backup adder is disabled by tristate buffers (RAS stands here for Redundant-Adder Shutdown). As mentioned before, ScAU refers to the scalable arithmetic unit, where '*nm*' stands for normal, full-width mode, and '*dm*' for downscaled mode.

From the synthesis results in the table, a number of conclusions can be drawn. As expected, the DAU and DAU-RAS show an increase in area and power consumption, compared to the single adder. However, the area and power consumption does not double because the input and output registers are not replicated; only the combinational logic is. The difference in area and power overhead between the single-adder AU and DAU is relatively small, which means that the sequential logic dominates both area and power consumption. When the unused adder is disabled, as in the DAU-RAS, the overall power consumption reduces significantly. Yet, disabling any of the two adders takes up more area due to the extra tri-state buffers needed.

Table IV. Synthesis results of 16-bit AUs (100 MHz, UMC 90 *nm* SP technology).

Design	Area	Delay	Power
	[units]	[ns]	[μW]
AU	1513	1.91	104.50
DAU	2185	2.09	181.60
DAU-RAS	2713	2.13	164.80
ScAU (nm)	2332	2.97	166.90
ScAU (dm)	–	–	118.90

Table V. Overheads compared to the single 16-bit AU (100 MHz, UMC 90 *nm* SP).

Design	Area	Delay	Power
	[%]	[%]	[%]
DAU	44.42	9.42	73.78
DAU-RAS	79.31	11.52	57.70
ScAU (nm)	54.13	55.50	59.71

The ScAU requires slightly more area than the DAU, but significantly less than the DAU-RAS. Even though the power consumption of the ScAU in normal mode is significantly lower than that of DAU, it is slightly higher than that of the DAU-RAS, which is power-optimized by disabling the backup adder. This is to be expected because, when the 16-bit backup adder in the DAU-RAS is disabled, there is no dynamic power consumed (only a small amount of static power is dissipated). Furthermore, there is not much additional hardware that consumes power apart from the output multiplexer and the tri-state buffers (32 tri-states in total, 16 of them in 'pass' state, 16 of them in 'HiZ' state) at the inputs. The ScAU on the other hand, does have the same power consumption for the adder part (2×8-bit adder segments $\equiv$ 16-bit adder), the same power consumption of tri-state buffers (also 32 tri-states in total, 16 of them in 'pass' state, 16 of them in 'HiZ' state), but has on top of that power dissipated by the additional control logic. This is the reason why the ScAU (normal mode) cannot dissipate less power than the DAU-RAS.

The power consumption of the ScAU in downscaled mode (dm) is significantly lower than in normal mode (nm), significantly lower than that of the DAUs, and only slightly higher than that of the single-adder AU. Unfortunately, the ScAU shows a large increase in delay because the additional logic resides in the adder's critical path. It will be shown later why this increase is irrelevant, since in the next design stage (when ED/EC mechanisms are added) the critical path will become shorter again.

Table V shows the area, delay and power overheads of the DAU, DAU-RAS and ScAU compared to the nonscalable, single-adder AU. What can be seen here is that the DAU provides fault tolerance by implementing a backup adder, however this comes at a large cost in terms of power (+73.78%) and area overhead (+44.42%). The DAU-RAS does the same, albeit the backup adder is disabled to save power. In that case the power overhead drops noticeably (+57.70%), yet the area cost almost doubles (+79.31%).

The ScAU incurs a power overhead similar to the DAU-RAS (+59.71%) but at a significantly lower area overhead (+54.13%), by comparison. Of course, it does so through a large increase in delay (+55.50%) which is still realistic for the class of applications we are aiming. It should be noted that, by adding an AU to the reference AU and all needed interconnecting logic, the ScAU incurs slightly more than 50% overheads across all three metrics, implying that the design is highly scalable with size in terms of costs. On the other hand, the DAU and DAU-RAS designs do not exhibit such uniform costs. If absolute numbers are compared (instead of overheads), the power consumption of the ScAU is 8.1% lower than the DAU and 1.3% higher than the DAU-RAS. The area of the ScAU is 6.7% higher than the DAU and 14.0% lower than the DAU-RAS.

Since the ScAU requires a significant amount of hardware to implement the scalability provisions, we decided to not examine a ScAU with four segments. This would require even more hardware and would only be efficient when the adder width is unrealistically large. For a 16-bit adder, where each of the four segments is no larger than only 4 bits, this will most certainly not be the case. For considerably wider architectures

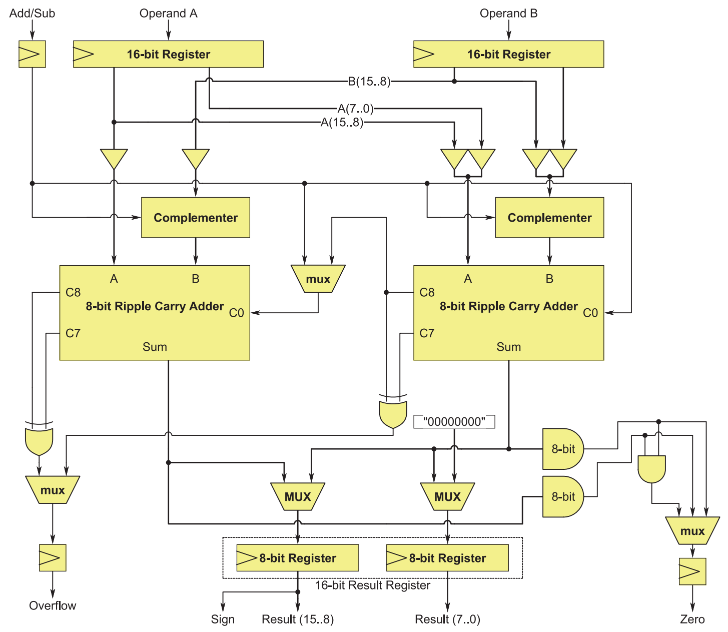

Fig. 2. Precision scalable arithmetic unit (2 × 8 bit RCA instance) with simplified design compared to the original ScAU. The zero-padding vector is applied to the output in case of error.

(such as 64 bits), 4 segments might be efficient but such data width is not practical for our targeted class of systems.

4.4. Precision-Scalable Arithmetic Unit

Another implementation of the ScAU which dramatically reduces the energy per instruction in downscaled mode is the *precision-scalable arithmetic unit* (PScAU), depicted in Figure 2. Instead of performing the addition in full precision over two cycles, some applications might tolerate discarding the lower (or higher) 8 bits of the operands. If an error occurs, the erroneous adder segment can be shut down and the work can continue with the remaining segment, where only the higher-order (or lower-order) byte of the result is computed and the lower-order (or higher-order) byte is padded with zeros.[1] In that case, the precision of the addition is compromised, but now only one cycle is required in downscaled mode instead of two. In Table VI, the synthesis results of the ScAU and the PScAU are compared (UMC 90 *nm*, at 100 MHz). In downscaled mode, the ScAU requires 42.5% more energy per operation (compared to normal mode) while the PScAU requires only 6.4% more energy. The PScAU is an option for low-power architectures wherein two-cycle operations are not permitted (e.g., due to unacceptable performance slowdown) but precision loss is acceptable.

Since the power consumption of the PScAU in downscaled mode is significantly less than in normal mode (−36.5%), the PScAU becomes interesting for dynamic thermal management. In systems where heat dissipation is a problem, the PScAU can temporarily switch to low-precision operations to limit the chip temperature. This might be very beneficial in the case, for instance, of implants whereby increased heat dissipation may lead to reduced device lifetime as well as tissue damage, fibrosis etc.

[1]Different padding policies might be explored each with a different precision-error margin.

Table VI. Comparison between ScAU and PScAU (100 MHz, UMC 90 *nm* SP technology).

Design parameter	ScAU	PScAU
Delay [ns]	2.97	2.14
Area [units]	2332	2050
Power (nm) [μW]	166.9	165.9
Power (dm) [μW]	118.9	106.40
Energy/instruction (nm) [pJ]	1.67	1.66
Energy/instruction (dm) [pJ]	2.38	1.06

Table VII. Overheads of 16-bit ScAU for different adder types with respect to the single 16-bit AU (100MHz, UMC 90 *nm* SP technology).

	Area	Delay	Power
ScAU design	[%]	[%]	[%]
ScAU (RCA-16/8)	54.1	55.5	59.7
ScAU (CSK-16/8)	49.7	88.4	55.6
ScAU (RCLA-16/8)	47.0	73.3	53.3

Henceforth, we will restrict our analysis to the ScAU since the PScAU is—computation-, resource- and fault-coverage-wise—a *proper subset* of the ScAU. In effect, if the ScAU proves better than the competition, then the PScAU will score even better than the ScAU in terms of the PDA metric; this can be seen in Table VI. Higher fault tolerance is also guaranteed by the simple fact that the PScAU contains a simpler design (thus, less logic gates) than the ScAU. Besides, using the full-precision ScAU is fairer when comparing with the reference designs such as the DAU-RAS.

4.5. Implementing Fast Adders in the ScAU

Table VII illustrates the overhead of the 16-bit ScAU with respect to the nonscalable, single-adder AU, when faster adder types are employed for the adder blocks inside the ScAU. The CSK and RCLA cases are studied, both being fast-adder designs, yet relatively cheap in terms of power and area. This study is performed for architectural scenarios wherein a RCA-based ScAU will not be fast enough, thus a fast-adder-based ScAU will be needed.

If a faster, larger adder is used, the impact of the ScAU's extra interconnection and bypass logic is expected to become less prominent. Indeed, as Table VII shows, the area overhead of the ScAU decreases with the adder size. Thus, the ScAU becomes overall more area-efficient when built using fast-adder blocks.

For delay, the opposite is true. The n-bit ScAU is composed of two $n/2$-bit adder blocks placed in series while the nonscalable AU employs one monolithic, n-bit adder block. Since these blocks are simple ripple structures, this segmentation is of no significance for RCAs but it is for the more complex fast adders. For instance, two $n/2$-bit CSKs in series require smaller carry-skip lines than a monolithic n-bit CSK and, thus, can never be as fast. Typically, fast adders become more efficient speed-wise when the word width increases. Therefore, the ScAU delay overhead increases when fast adders are employed.

Finally, the ScAU power overhead decreases when fast adders are used. This phenomenon can be explained in the exact same way as the decrease in area overhead: fast adders utilize more power, which makes the contribution of the overall logic less significant. In conclusion, it can be said that, for higher adder speeds, fast-adder building blocks make the ScAU more efficient since they reduce both area and power.

5. SCAU ED/EC ANALYSIS

Having facilitated functionality segmentation in the ScAU, in order to add fault tolerance, we have to retrofit the ScAU with ED and EC techniques next. The EC technique for the ScAU has already been discussed: when an error occurs in one of the adder segments, the particular segment is shut down and the ScAU continues with the computations utilizing the remaining adder segment. The calculations, then, require two clock cycles (ScAU) or a single cycle but with lower precision (PScAU). Earlier, the datapath and scalability features of the ScAU were presented. In Section 6, an EC controller responsible for reconfiguring the ScAU in the presence of errors will be introduced. In this section, we focus on suitable ED schemes to trigger that controller. A number of well-known ED schemes is discussed and compared with low-power and low-area constraints in mind.

As mentioned in Sellers et al. [1968] and Davis [1965], single errors occur much more often than double errors.[2] In fact, the fault coverage does not increase significantly when the ED scheme is extended for multiple errors. What does increase quite dramatically is the cost incurred in the design by the ED scheme. On the other hand, the chance of double errors is small (but certainly not zero). Ideally, we would like to check for all possible errors but, given the tight resource constraints of embedded systems, we have based our choice of an ED scheme on a trade-off between reliability levels and implementation costs. Besides, significant work has already been done in the field of high-performance, resource-relaxed systems [Kumar and Aggarwal 2006; Purohit et al. 2010; Slayman 2005]. Therefore, we focus on both error coverage and costs of the following schemes:

—Berger check prediction [Gorshe and Bose 1996; Lo et al. 1992];
—Bose-Lin check prediction [Gorshe and Bose 1996; Mitra and McCluskey 2000];
—parity (or modulo-2) checking [Sellers et al. 1968; Hsiao and Sellers 1963]; and
—residue (or modulo-3) checking [Sellers et al. 1968; Langdon and Tang 1970].

Whether an ED scheme is eligible for implementation in the ScAU depends on two factors: first, its *cost* should be acceptable, and second, it should be compatible with the *scalability* property. Since the ScAU is subdivided into two adder segments (placed sequentially during normal operation mode), an identical instance of the ED scheme can be applied to each of the segments independently. However, achieving fault secureness for single errors in the ScAU as a whole is difficult in case of two independent instances of ED, as will be explained later (see Section 6.4).

5.1. Berger and Bose-Lin Check Prediction

Berger check prediction (BCP) offers a high fault coverage, since it covers single and double arithmetic errors, as well as all unidirectional errors.[3] However, BCP is not suitable for implementation in low-power and low-area embedded systems, since the area cost is at least close to that of full hardware duplication (if not costlier).

Besides, the area cost of Bose-Lin check prediction (BLCP) is reported to be (prohibitively) high, but [Mitra and McCluskey 2000] do not mention how many code bits they utilize. However, since [Gorshe and Bose 1996] report that in a 16-bit adder with 2 Bose-Lin code bits (capable of detecting single arithmetic errors, and double unidirectional errors) the area cost is significantly lower than that of utilizing BCP, BLCP (with a limited number of code bits) ought to be cheaper than full hardware duplication. Implementation and analysis are required to provide an answer, since it cannot be found in the literature. It is highly unlikely, though, that BLCP will ever be cheaper than parity checking (to be discussed next). After all, unlike parity checking, BLCP is designed for detecting multiple errors and requires at least two check bits. Still, BLCP might be an interesting technique when one requires a (slightly) higher error coverage than parity or residue checking.[4]

[2]A single error causes a faulty single-bit value. Multiple errors are two or more single errors manifesting simultaneously. Double errors are the particular case of two single errors.
[3]Unidirectional errors cause either ones to flip into zeroes or zeroes into ones. Such errors typically originate in asymmetric communication channels.
[4]Further research could elucidate the exact costs of Bose-Lin coding in comparison to other ED schemes.

5.2. Parity and Residue Checking

Having dismissed the Berger and Bose-Lin check-prediction schemes, we move now to the parity- and residue-checking techniques. Based on the literature, both seem interesting ED techniques for implementation in the ScAU. The fault coverage of the schemes is identical: both schemes are fault-secure for single errors and cover 50% of all double errors (not necessarily the same 50%). However, it is difficult to decide which technique is most suitable for integration in the ScAU. In the literature, parity checking is mentioned as cheaper than residue checking, but Langdon and Tang [1970] prove otherwise, under certain conditions.

According to Mitra and McCluskey [2000], residue checking is never economical unless the operands are already provided along with the residue check bits. This is a consequence of the generation of check bits: generating residue check bits (for prediction) is considerably more expensive than generating parity bits [Langdon and Tang 1970; Mitra and McCluskey 2000]. However, Langdon and Tang [1970] agree that residue checking is cheaper than parity checking as long as the operands are already provided along with the check bits.

A closer look shows that Langdon and Tang [1970] have utilized a CLA and a high-speed, parity-prediction circuit. The circuit is employed to avoid unnecessary delays by the ED logic [Sellers et al. 1968]. Since the predicted parity is available considerably later than the result, this affects the adder's throughput. Ideally, the result and the predicted parity should be available at the same moment in time. However, high-speed parity prediction increases the costs of the ED scheme significantly. On the contrary, if high performance is not required, standard-speed parity prediction can be implemented in which case residue checking becomes more costly. Langdon and Tang [1970] do not explicitly mention this.

As already mentioned, generating residues inside the AU is not efficient. This implies that there must be reuse for the residue check bits for other error-checking purposes in the architecture as well, apart from the AU. When an architecture contains, for example, multiple adders/subtracters, a multiplier and a divider, residue checking can preferably be employed to check all these units. Residue checking can also be employed to protect data transfers and memories. It is possible to modify the residue checker to check logical operations as well, but this requires additional hardware. What is more, this approach makes the separation of the AU and logical unit (LU) impossible. This is undesirable if long sequences of logical operations occur frequently. Moreover, checking memories and data transfers by parity prediction is more efficient since only one parity bit per data word is required.

To summarize, residue checking can be cheaper than parity checking for larger, fast adders (provided that the check bits are already present and that the architecture can use the check bits for more purposes than just the adder, for instance, when also a multiplier is present in the architecture). However, in most situations parity checking is to be preferred over residue checking, when the two major drawbacks of the scheme mentioned above are considered. For this reason, a residue-checked AU was abandoned in this study; a *parity-prediction scheme* has been implemented and compared against the previously mentioned hardware-replication techniques.

In general, if a higher degree of error detection is required (higher than fault-secure for single errors and 50% coverage of double errors), residue checking becomes attractive. Note that area and power grow significantly when the modulus is increased (the overhead scales linearly with the amount of check bits utilized). We believe, however, that despite their high cost, the very high fault coverage of the TMR or QMR(-RAS) schemes would probably make them better choices, when a higher degree of error detection is demanded.

5.3. Parity-Prediction-Scheme Comparison

There exist several parity-prediction designs which are fault-secure for single errors. Two of them will be discussed here. These two schemes are chosen since they bear the highest potential in terms of low power consumption, low-area, and reliability.

5.3.1. Duplicated-Carry Scheme. Any parity-prediction scheme for adders contains the following parts:

—parity (P_A and P_B) generators for the operands (XOR-trees);
—parity (P_C) generator for the internal carries;
—parity (P_S) generator for the parity of the sum; and
—predictor/comparator, which requires P_A, P_B, and P_C to predict P_S and compare it with the real P_S, and signal an error on mismatch.

As established in Section 3.4, carry errors always manifest as multiple errors of even number. Therefore, the parity-prediction scheme would normally not be able to detect these types. [Sellers et al. 1968] proposed a parity-prediction scheme with duplicated-carry circuits which is fault-secure and is also covering all carry errors.

When the carry circuit of every full-adder is duplicated, it is possible to compare the carry outputs of the full adder and the duplicated-carry circuit. Since the aim is fault secureness for single errors, only one carry circuit is assumed to be erroneous at a time. When the compare signals are fed to the checker, we can detect single carry errors since the maximum number of detected carry-circuit errors is one, and, thus, odd. [Sellers et al. 1968] called this scheme “Duplicate carry with parity check I”.

In fact, the scheme can be simplified by avoiding the actual comparison between the ‘normal’ carries and duplicated carries. It is sufficient to generate parity P_C based on the duplicated carries (instead of the normal ones) in order to achieve the fault-secure property. In case of a carry error, the parity of the duplicated carry $P_{C,d}$ always contains one error less than P_C. Furthermore, the number of errors in the sum S is always equal to the number of errors in P_C (see Section 3.4). This makes the error detectable. This scheme is called “Duplicate carry with parity check II”, Figure 3. Note that the P_A/P_B parity generators are not shown. The ODD circuit implements the checker, P_S, and P_C generator all in one. The implementation is an n-input XOR; if the number of 1’s at the inputs is even, the output is 0, otherwise it is 1.

5.3.2. Carry-Dependent Sum Adder Scheme. The parity-prediction scheme based on the carry-dependent sum adder (CDSA) was proposed by [Hsiao and Sellers 1963; Sellers et al. 1968]. The most elementary distinction with the previous scheme is that the carry circuits are not duplicated. If an error in carry c_n causes an error in c_{n+q}, then it also causes an error in subsequent $c_{n+1}, c_{n+2}, \ldots, c_{n+q-1}$. All carry errors lead to (successive) erroneous sum results as well: the results $s_{n+1}, \ldots, s_{n+q+1}$ form an error too. Since carry errors always cause an even number of errors (which makes them undetectable), in the CDSA scheme the full-adder cells are modified in such a way that this is no longer the case. The idea is to make sure that when the error occurs in c_n, also s_n is erroneous. If that can be done, $s_n, \ldots, s_{n+q+1}$ is also faulty, having one error more than before, turning the total amount of sum and carry errors into an odd number. A carry-dependent sum full-adder cell is depicted in Figure 4. The checker can be implemented by an ODD circuit, similar to the one shown in Figure 3. A complete implementation of the CDSA scheme is shown in Figure 7.

5.4. ScAU ED-Scheme Selection

The choice between duplicated-carry and CDSA schemes will be based on costs, since the fault coverage of both schemes is equal. In the CDSA scheme, the full-adder cells

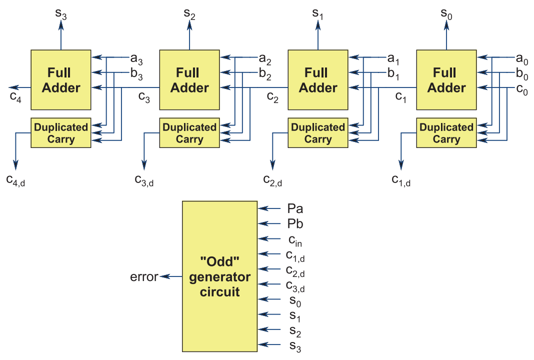

Fig. 3. Adder with duplicated carries - Addendum II [Sellers et al. 1968]. Top: 4-bit adder chain with carry-duplication modules. Bottom: Input-parity, duplicated-carry and sum bits are fed to an ODD detector circuit which outputs ‘1’ on error.

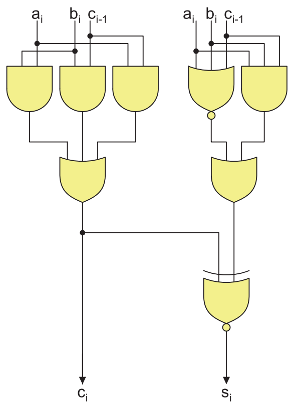

Fig. 4. Single-bit slice of carry-dependent sum full-adder [Hsiao and Sellers 1963].

are more complex, but the carry circuits are no longer duplicated. However, it is to be expected that CDSA costs are lower. In the literature, however, no comparison between the two parity schemes could be found. Therefore, both schemes were implemented, synthesized and carefully analyzed.

Synthesis provides us with the expected information: the parity-prediction scheme based on the CDSA is cheaper than the one based on duplicated carries. The CDSA scheme requires 5.0% less area and 5.3% less power. The CDSA scheme does, however, have a longer delay than the duplicated-carry scheme. This is because the carry-dependent full-adders are more complex and, consequently, slightly slower. For example, the 16-bit PC-DAU-RAS design implemented with the duplicated-carry scheme is 11.5% faster than the same design implemented with CDSA.

It depends on the targeted application which ED scheme is best-suited for the ScAU. We intend to employ the ScAU in the SiMS (Smart Implantable Medical Systems) micro architecture. SiMS is a research project aiming at biomedical microelectronic devices fully implanted to the human body. Obviously, within the SiMS context reliability is crucial. On the other hand, as explained in [Strydis et al. 2006], the SiMS micro-architecture is extremely minimalistic and power and area budgets are very tight. Oftentimes, increasing reliability might lead to intolerable power and area. So, also for highly mission-critical devices such as biomedical implants, there is a trade-off between reliability and cost.

We consider BCP and BLCP not suitable for ScAU implementation, because of the high costs. Also, residue checking is not particularly interesting: the SiMS architecture has only one AU, and no multiplier or divider. Thus, apart from the ScAU, there is no reuse for the residue check bits, making it inefficient.

The decision was made to implement CDSA parity checking and have the design fault-secure for single errors only. We believe this is justifiable because—as established earlier—single errors occur much more frequently than errors of higher multiplicities. The extra protection against double or even triple errors does not add significantly to the probability of detecting errors. What is more, to cope with the chance of multiple errors within the SiMS context, the intention is to periodically run online tests with a set of customized test vectors to check the AU [Seepers et al. 2012]. This way, some multiple errors may not be covered by the ED-scheme, but ultimately they will be captured by the on-line testing.

The details about how the on-line self-tests are implemented in the SiMS microarchitecture can be found in Seepers et al. [2012]. On-line testing is performed in order to detect soft faults by executing the same instruction twice and comparing the results. If the results differ, the pipeline is flushed (thus, no faulty instructions are committed). This testing method complements the ScAU. The ScAU is designed in such a way that it does not discriminate between hard and soft faults. Since soft faults occur far more often than hard faults, it would be inefficient to shut down a segment of the ScAU based on a single occurring soft fault. In order to detect hard faults, an architecture-specific sequence of instructions is executed and the results are compared to an expected value. If a mismatch is found, it can be determined which part of the architecture (e.g., one of the ScAU segments) has succumbed to hard faults, based on the mismatched value. This value is prestored in a register at compile-time but it may as well be stored elsewhere, for instance, in the data memory.

6. PARITY-CHECKED SCAU

In the previous section we selected the most suitable ED scheme for our ScAU. We can, now, introduce the *Parity-Checked ScAU* (PC-ScAU), an extension to the ScAU design.

6.1. Implementation of the PC-ScAU

In the PC-ScAU, both AU segments are equipped with parity prediction and an EC-scheme is implemented, which disables the damaged AU segment when an error is detected, downscales the structure, and proceeds with the computation using the remaining AU segment. We assume that the parity bits are generated at an early stage in the pipeline, like it was assumed for the PC-DAU-RAS. Parity bits generated for 16-bit operands are useless when the decision is made to check the upper and lower byte independently. So, either (i) two check digits per operand must be generated (one for the lower-order byte P_{low} and one for the higher-order byte P_{high}), or (ii) P_{low} and P_{high} must be derived from the 16-bit check digit P.

For option (i), the so-called split-parity, P_{low} and P_{high} are generated early in the pipeline and propagated all the way to the AU stage, which means that each pipeline register must have extra room for two parity bits instead of one. Parity checkers in the processor needing the parity of the full 16-bit operand P, can easily compute it by XOR-ing P_{low} and P_{high} which results in extra overhead across various pipeline stages.

Option (ii), where P_{low} and P_{high} are derived from P, would require an eight-bit parity generator placed locally in the AU stage to generate P_{low}. Then, P_{high} can be computed by XOR-ing P and P_{low}. Also, extra overhead is present in this case but three flip-flops (two at the input, one at the output) can be saved since the pipeline registers are incorporated in the PC-ScAU design. This solution increases AU costs.

Implementation and synthesis of both options has verified that option (ii) is indeed slightly more expensive in terms of power (+0.8%) and area (+3.7%), from the AU standpoint. Although option (i) is more expensive when the complete pipeline is considered, we have decided to implement—without loss of generality—option (i) here for maintaining fairness in our comparisons with the rest of the AU designs[5].

The schematics of the PC-ScAU design are depicted in Figure 5. All components related to ED/EC are displayed in green (dark gray when seen in grayscale). The split-parity bits at the inputs are fed to multiplexers. In case a segment is shut down, the parity bits can be rerouted this way. $P_{r,h}$ and $P_{r,l}$ represent the split-parity bits of the result. The FSM of the EC-controller (error-correcting controller) is depicted in Figure 6. The main task of the EC-controller is to reconfigure (downscale) the PC-ScAU when it receives an error notification from one of the parity checkers. The cycle-controller

[5]Further research is needed to find the cost of option (i) for the pipeline of different processor organizations.

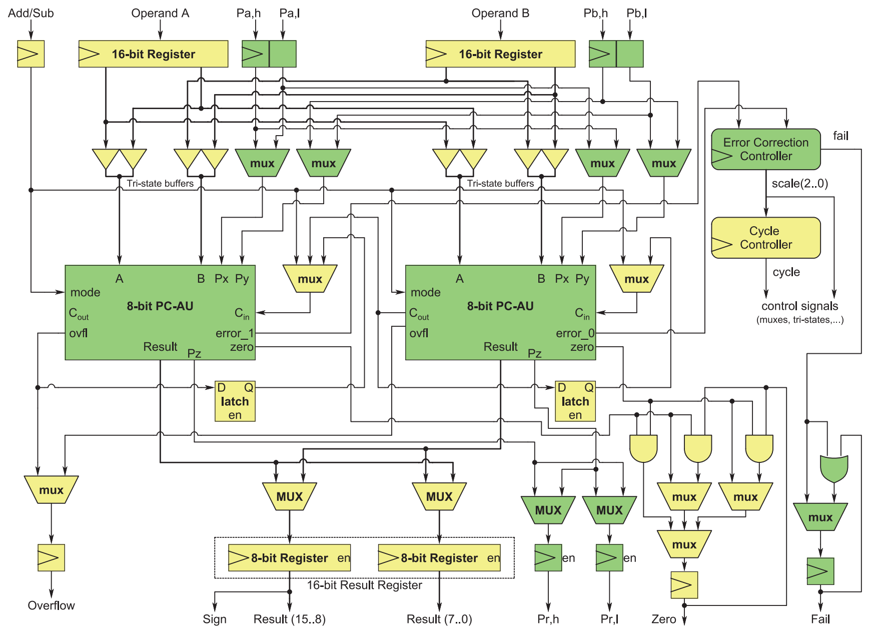

Fig. 5. The PC-ScAU (2×8-bit instance) with ED/EC enhancements highlighted.

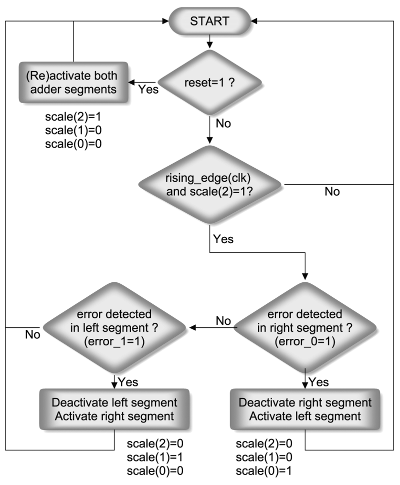
START
(Re)activate both adder segments
Yes
reset=1 ?
No
scale(2)=1
scale(1)=0
scale(0)=0
rising_edge(clk) and scale(2)=1?
No
Yes
error detected in left segment ? (error_1=1)
error detected in right segment ? (error_0=1)
No
Yes
Deactivate left segment Activate right segment
Deactivate right segment Activate left segment
scale(2)=0
scale(1)=1
scale(0)=0
scale(2)=0
scale(1)=0
scale(0)=1

Fig. 6. FSM of the EC-controller.

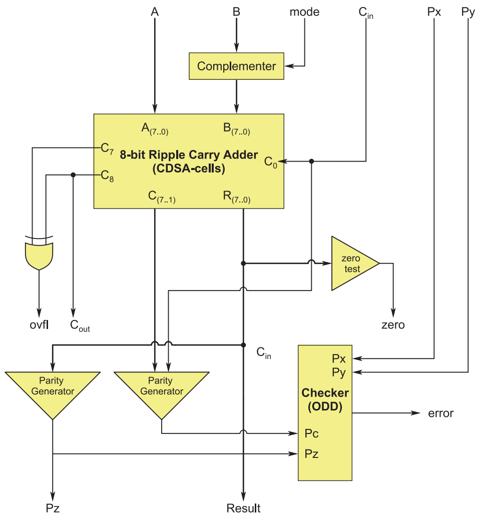

Fig. 7. The parity-checked AU (CDSA scheme).

(which keeps track of the current cycle) will be notified depending on whether and how the ScAU is reconfigured. Both the EC-controller and the cycle-controller steer the various multiplexers, tri-state buffers, etc. in the design. The internal circuitry of the 8-bit parity-checked segments is depicted in Figure 7. P_x and P_y represent the input parity bits and P_z the output parity. Note that the characteristics of the checker in principle does not differ from the parity generators (generating P_z and P_c): those are all XOR trees called ODD circuits.

Table VIII. Synthesis results of AU for different ED/EC schemes. Best candidates per result category are in bold.

ED/EC scheme	Area	Total Power	Dynamic Power	Static Power	Delay
	[units]	[μW]	[μW]	[μW]	[ns]
TMR	3,380	51.48	47.88	3.60	3.67
QMR	3,842	58.76	54.38	4.38	**3.34**
QMR-RAS	4,308	45.88	40.97	4.91	3.73
PC-DAU-RAS	3,435	**37.88**	**34.18**	3.70	4.75
PC-ScAU (nm)	**2,884**	40.64	34.47	**3.17**	4.18
PC-ScAU (dm)	*2,884*	*29.22*	*26.00*	*3.22*	*<4.18*

6.2. Implementation of Reference Designs

Our fault-tolerant PC-ScAU will be compared to a number of fault-tolerant AUs:

—AU with Triple Modular Redundancy (TMR);
—AU with Quadruple Modular Redundancy (QMR);
—AU with QMR, with redundant adder shutdown (QMR-RAS); and
—duplicated AU with parity check and redundant adder shutdown (PC-DAU-RAS).

As established earlier, the QMR and TMR have a high fault coverage (capable of detecting all errors of any multiplicity, except common-mode errors) but very high costs. Even though QMR and TMR schemes are typically too costly for implementation in a micro-architecture with very low-power and low-area budgets, the decision was made to implement these schemes for a number of reasons. One reason is to use them as reference points for comparisons with other ED/EC designs. Another less obvious reason for including the QMR is because the typical design can be optimized for lower power consumption and, thus, offer an additional interesting comparison with the proposed PC-ScAU. As explained in Section 3, the QMR contains two Self-Checking Adders (SCAs), both simultaneously active. There is, however, no need for both SCAs to be active simultaneously. The design is, then, modified in such a way that only one SCA is active at a given time. When the active SCA detects an error, it is shut down and the nonactive SCA is activated. This modification will increase the scheme area but we are primarily interested in its power savings. Note that this modified QMR scheme (QMR-RAS), is no longer an error-masking scheme. Since the failing component is shut down, a new component is activated and the data path is reconfigured, this scheme is now based on repair/reconfiguration.

The last and most important reference design included in our comparisons is the DAU-RAS. This design is built with two AUs, each checked by CDSA-based parity prediction. Also, this design is power-optimized (based on repair/reconfiguration): only one AU is active at a time (thus the RAS suffix). The ScAU is also checked by parity prediction based on the CDSA scheme, now called the PC-ScAU.

6.3. Synthesis Results and Evaluation

After describing all designs in VHDL, they were synthesized and analyzed for area, power and delay at 20 MHz which is a realistic system frequency for the SiMS architecture. The SiMS architecture is 16-bit wide; therefore, without loss of generality, we implemented 16-bit instances of the ScAU and the reference designs. Based on our previous analysis in Section 3.3, the RCA appears to be the most suitable adder for our specific purpose and its speed in 90-*nm* CMOS will most certainly be high enough for SiMS. The synthesis results are shown in Table VIII and Figure 8.

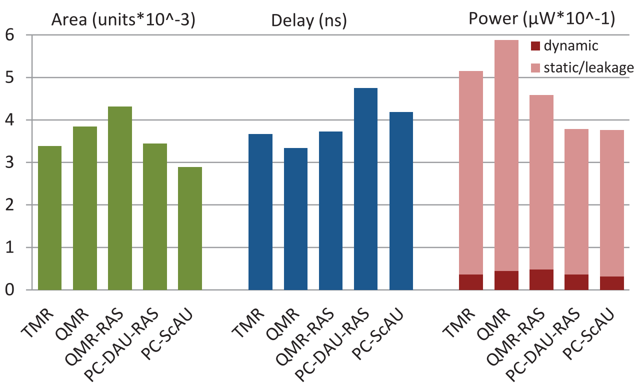

Fig. 8. The area, delay and power requirements of the AUs for different ED/EC schemes.

Figure 8 reveals the QMR and TMR schemes to perform far worse than the rest when power and area costs are of interest. These schemes are, however, the fastest. The QMR-RAS, optimized for power, does indeed result in significant power savings (−22%) over QMR. This makes the scheme, in terms of power, even more attractive than the TMR. The area cost is, however, nonnegligible (12% higher compared to QMR, which had a very high area cost to begin with). Even so, for scenarios that require multiple-error detection at a relatively low power cost, the QMR-RAS is a suitable candidate, as long as the high area cost can be tolerated. For the currently targeted architectures, allowing very limited power and area budgets, the QMR-RAS is obviously not an option. Thus, resorting to cheaper error-detection codes is necessary when area budgets are very tight, since all ED-schemes with full hardware replication (QMR, TMR) incur high area costs.

The PC-ScAU shows interesting trends. Its critical-path delay is shorter than that of the PC-DAU-RAS (−12%), while in Section 4.3, (without ED/EC provisions) exactly the opposite was the case. Two are the suspected reasons for this counter-intuitive phenomenon: First, the path through the ED logic is shorter in the PC-ScAU than it is in the PC-DAU-RAS. The critical path goes through the first 8-bit adder segment, then—via the carry-out, connecting logic, and carry-in—through the second 8-bit adder segment and, finally, through the 8-bit parity tree of the sum of the second adder segment. In the PC-DAU-RAS case, the critical path goes through the 16-bit adder and then through the 16-bit parity tree of the sum. Obviously, an 8-bit parity tree is faster than a 16-bit one. Second, the critical path of the PC-DAU-RAS was altered by adding tri-state buffers for input-gating the add/sub signal. The adder and complementer cannot perform any useful computations until the add/sub signal has arrived. Since there is a tri-state buffer present on this path (controlled by the EC-controller), the path is somewhat lengthened. To summarize, the critical path of the PC-ScAU has not been lengthened to the same extent as the PC-DAU-RAS by the ED logic, and the critical path of the PC-DAU-RAS has become longer due to input-gating the add/sub signal.

On the other hand, the PC-ScAU performs marginally worse in terms of power than the PC-DAU-RAS (+7%). In Section 4.3, we concluded that the ScAU and DAU power consumptions (without ED/EC) were very similar at 20MHz. The reason why the PC-ScAU has a higher power consumption than the PC-DAU-RAS is because of the additional multiplexers (multiplexing the split-parity bits), the additional split-parity bit registers, and the added EC-controller in the PC-ScAU. However, it is noteworthy that, while the dynamic-power component of the PCScAU is marginally higher than that

Table IX. Ranking of AUs according to PDA-variant cost metrics. Best candidates are in bold.

AU	PA **x**10^5	PD **x**10^2	AD **x**10^4	PDA **x**10^5	P^2DA **x**10^7	PD^2A **x**10^6	PDA^2 **x**10^9	P^3DA **x**10^9	PD^3A **x**10^7	PDA^3 **x**10^{13}
TMR	1.74	1.89	1.24	6.39	3.29	2.34	2.16	1.69	0.86	0.73
QMR	2.26	1.96	1.28	7.54	4.43	2.52	2.90	2.60	**0.84**	1.11
QMR-RAS	1.98	1.71	1.61	7.37	3.38	2.75	3.18	1.55	1.03	1.37
PC-DAU-RAS	1.30	1.80	1.63	6.18	2.34	2.94	2.12	0.89	1.39	0.73
PC-ScAU	**1.17**	**1.70**	**1.21**	**4.90**	**1.99**	**2.05**	**1.41**	**0.81**	0.86	**0.41**

Table X. Qualitative comparison between different ED/EC schemes.

ED/EC scheme	Area	Latency/ operation	FT	Instant Power	Energy/ operation
TMR	□/+	+	++	−	−
QMR	−	++	++	−−	−−
QMR-RAS	−−	+	++	□	□
PC-DAU-RAS	□/+	−	□	++	++
PC-ScAU (nm)	++	□	−/□	+	+
PC-ScAU (dm)	++	−−	−/□	++	−−

Legend: ++ represents the best, + a good, □ a moderate, − a relatively poor, and −− the least attractive implementation according to a certain criterion.

of the PC-DAU-RAS (+1%), the static-power component is significantly lower (−14%), as can be seen in Table VIII. This means that the dynamic component dominates the static one, thus the net increase in power of the PC-ScAU. However, this also means that (i) for systems with longer standby periods, the PC-ScAU may well exhibit lower power needs, and (ii) employing newer technology nodes (e.g. 25 *nm*) which show comparable dynamic and static power components will, most probably, give the PC-ScAU a lower total power budget over the PC-DAU-RAS (and the rest of the reference designs).

Regarding area, the PC-ScAU requires the least amount of transistors, compared to all ED/EC implementations. So, even though the PC-ScAU performs slightly worse in terms of power compared to the PC-DAU-RAS, it consumes 16% less area (Table VIII).

Conclusively, the PC-ScAU design behaves excellent in terms of delay and area while consuming marginally more power than other approaches at 90 *nm*. As an overview, it is interesting to validate its efficiency with respect to various PDA-derived metrics. In Table IX, various established PDA variants are being reported. The first four (columns 1 to 4) are standard with all appearing objectives (power, area and delay) assigned an equal weight. It depends on the target application whether, for instance, power savings are of greater significance than area savings. Therefore, to offer more insights on the different adder designs, extra PDA-variants with higher powers assigned to power, delay and area are included (columns 5 to 10). Provided that the fault-coverage level offered by the PC-ScAU is sufficient, we can observe from Table IX that the PC-ScAU is the best-performing design against all other alternatives in terms of PDA and other variants. There is a single exception: when delay is raised to the third power (PD^3A), the PC-ScAU ranks second (together with TMR), losing only to QMR (the fastest design in our study). Even then, though, the difference is marginal.

As a final contribution, a qualitative overview of all the implemented and studied fault-tolerant AUs has been compiled in Table X. The table illustrates in a graphical way all the findings of this work and can be used as a practical 'cheat sheet' by fault-tolerant-AU designers. It essentially shows that the PC-ScAU is the most balanced design in terms of resources compared to the reference AUs.

6.4. Fault Secureness of the PC-ScAU

As mentioned before, the parity scheme selected for the ScAU makes its two separate 8-bit adder blocks fault-secure for single errors. Nevertheless, when connected together forming the 16-bit scalable adder, the two 8-bit blocks are not fault-secure for single errors as observed during the design of the PC-ScAU. This is caused by a small number of so-called ‘weak spots’ in the error-detection scheme: logic parts along the data path that are not checked for errors. The parity-prediction scheme assumes that the carry input of the adders is correct. However, the carry input (which is the add/subtract signal) is not connected directly to the carry inputs of the two 8-bit adders. Because of the scalable nature of the scheme, the carry inputs have additional multiplexers and latches attached to these inputs. When, for example, a stuck-at fault occurs at the output of such a multiplexer or latch, the carry input is erroneous and also the computation will be incorrect, which is not detected by the ED scheme. There are two methods to overcome this problem.

(1) Since the self-testing property of the 16-bit adder is not compromised by the additional logic, an error is by definition still detectable. In the SiMS micro-architecture, sufficient idle time is available to test the AU on a regular basis using a predetermined set of test vectors to test the entire ScAU for errors, as mentioned earlier, in Section 5.4.
(2) The logic components that are not covered by the parity-prediction scheme can be duplicated and the outputs can, for instance, be compared using a simple XOR gate. The outputs of the XOR gates can be considered as additional error signals, which should be represented by the existing error signal. Obviously, this comes at a (small) price. It is estimated that area will increase by 60 units (2.1%) and power by 0.5 μW (1.2%). By implementing this method, the fault-secure property for single errors of the 16-bit ScAU is achieved.

Which method is viable ultimately depends on the demands of the application and the exact power/area budgets. If an incorrect result is intolerable at all times, method (2) is the only legitimate option (if higher cost can be tolerated), since method (1) might pass incorrect results in between self-tests. We decided to opt for method (1). The reason for this is that the unchecked logic that forms the ‘weak spots’ in the ED is only a tiny fraction of the overall logic. Statistically speaking, the chance that an error occurs in these parts is small. Further, an additional increase in power consumption of the PC-ScAU would be highly undesirable. An additional 0.5 μW will increase the gap in power consumption between the PC-DAU-RAS and the PC-ScAU further.

6.5. Power Consumption of the PC-ScAU

As shown before, the PC-ScAU is capable of significant area and delay savings when compared to the PC-DAU-RAS. The power usage of the PC-ScAU and PC-DAU-RAS are very close with the PC-ScAU consuming slightly more. There are, however, situations where the power consumption of the PC-ScAU might become lower than that of the PC-DAU-RAS.

(1) When the adder is wide (e.g., 32 bits). While the adder itself scales linearly with the word size, the largest part of the control logic of the PC-ScAU remains unchanged. Thus, the overhead of the PC-ScAU becomes proportionally lower.
(2) When the adder needs to operate at high frequencies or a high throughput is desired. Then, a faster adder is required (e.g., CLA). Since fast adders are larger and consume more power, the overhead of the ScAU lowers by proportion.
(3) The total-power numbers obtained for the PC-ScAU and the PC-DAU-RAS are close. However, the PC-ScAU is considerably smaller in size than that the PC-DAU-RAS

and also exhibits a much smaller static-power component. Due to these two reasons, it is very well possible that, for a more recent and low-power technology node, the PC-ScAU will outperform the PC-DAU-RAS in terms of power.

All three points indicate that the 16-bit, RCA-based instance of the PC-ScAU used in our evaluation is, in many ways, a worst-case design point. By implementing a wider datapath, a faster adder and/or a more recent technology node in the future, the PC-ScAU family of adders is expected to narrow or even bridge the power gap to the other designs.

7. CONCLUSIONS

Instead of coarse-grain hardware duplication, we presented a novel design of an arithmetic unit (AU) containing a single adder with scalable size. The goal of this approach is to save power and/or area compared to regular, nonscalable, fault-tolerant designs. In normal mode, the ScAU performs one operation per cycle and utilizes the full adder width. The adder itself is divided into two segments. Once an error has been detected in one segment, the ScAU will disable it, reconfigure the data path, and continue with the remaining segment. When in such mode, the ScAU cycle count will change from one to two cycles per operation. The throughput will be compromised, however the computational precision will be preserved. A Precision-Scalable variant of the ScAU (PScAU) was, also, introduced in order to provide a solution for systems where the AU throughput is critical. The design is similar, however, in case of an error, the precision is now halved in order to preserve the throughput. After downscaling the PScAU, it will still be able to perform calculations within a single cycle, but with a reduced precision. The PScAU is also interesting for dynamic thermal management. Since the power consumption in downscaled mode is lower, one could decide to lower precision once critical chip temperature is approached.

A number of error-detection (ED) schemes were explored to investigate whether they are applicable within low-power and low-area embedded systems. A thorough comparison between residue checking and two types of parity checking was presented. In combination with regular self-checks at the software level, the decision was made to implement parity checking based on the Carry-Dependent-Sum-Adder scheme.

The ScAU, PCScAU, PScAU and several reference designs have been implemented, synthesized, and characterized in 90-*nm* UMC technology. The power consumption of the parity-checked ScAU (PC-ScAU) is low and we gain significant area savings compared to the cheapest reference design (a fault-tolerant arithmetic unit based on hardware replication, utilizing the same ED scheme). The PC-ScAU has a slightly higher power consumption (+7%), but a significant lower area (−16%) and delay (−12%), which results in a PDA product that is almost 21% better. Overall, the PC-ScAU ranks first in 9 out of 10 PDA-variant metrics, second to QMR only in the case of PD^3A.

ACKNOWLEDGMENTS

We would like to acknowledge Daniele Ludovici, Ioannis Sourdis, George Stefanakis, and Arjan van Genderen for their precious help.

REFERENCES

R. Amirtharajah, Thucydides Xanthopoulos, and A. Chandrakasan. 1999. Power scalable processing using distributed arithmetic. In *Proceedings of the International Symposium on Low-Power Electronics and Design*. ACM, 170–175.

S. Borkar. 2005. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro* 25, 6, 10–16.

T. K. Callaway and E. E. Swartzlander. 1992. Optimizing arithmetic elements for signal processing. In *Proceedings of the Workshop on VLSI Sig. Processing*. 91–100.

G.-C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano. 2006. Fault Localization, error correction, and graceful degradation in Radix 2 signed digit-based adders. *IEEE Trans. Comput.* 55, 534–540.

R. A. Davis. 1965. A checking arithmetic unit. In *Proceedings of the American Federation of Information Processing Societies National Semi-Annual Computer Conference*. ACM, 705–713.

S. S. Gorshe and B. Bose. 1996. A self-checking ALU design with efficient codes. In *Proceedings of the IEEE VLSI Test Symposium.* 157–161.

P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar. 2003. Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-μm to 90-nm generation. In *IEEE International Electron Devices Meeting* (*IEDM'03) Technical Digest*. 21.5.1–21.5.4.

J. L. Hennessy and D. A. Patterson. 2012. *Computer Architecture: A Quantitative Approach* Morgan Kaufmann Publishers Inc.

M. Y. Hsiao and F. F. Sellers. 1963. The carry-dependent sum adder. *IEEE Trans. Electronic Computers* 12, 3, 265–268.

ITRS. 2011. International Technology Roadmap for Semiconductors. www.itrs.net/Links/2011ITRS/Home 2011.htm. (2011).

A. Iyer and D. Marculescu. 2001. Power aware microarchitecture resource scaling. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe* (*DATE'01*). IEEE, 190–196.

S. Kaxiras. 2008. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers.

C. Khedhiri, M. Karmani, B. Hamdi, and K. L. Man. 2012. A fault tolerant adder based on alternative computation. *Int. J. Design, Analysis Tools Integrated Circuits Systems* 3, 14–18.

S. Kumar and A. Aggarwal. 2006. Reducing resource redundancy for concurrent error detection techniques in high performance microprocessors. In *Proceedings of the International Symposium on High-Performance Computer Architecture* (*HPCA'06)*. 212–221.

Sumeet Kumar, Prateek Pujara, and Aneesh Aggarwal. 2005. Bit-sliced datapath for energy-efficient high performance microprocessors. In *Power-Aware Computer Systems,* Lecture Notes in Computer Science, vol. 3471, Springer, 30–45.

Eren Kursun, Glenn Reinman, Suleyman Sair, Anahita Shayesteh, and Tim Sherwood. 2005. Low-overhead core swapping for thermal management. In *Power-Aware Computer Systems,* Lecture Notes in Computer Science, vol. 3471, Springer, 46–60.

G. G. Langdon and C. K. Tang. 1970. Concurrent error detection for group look-ahead binary adders. *IBM J. Res. Dev.* 14, 5, 563–573.

H. Lee. 2005. Power-aware scalable pipelined booth multiplier. *IEICE Trans.* 88, 11, 3230–3234.

Ting-Wei Lin, Ming-Chung Lee, Fang-Ju Lin, and Herming Chiueh. 2005. A low power ALU cluster design for media streaming architecture. In *Proceedings of the 48th Midwest Symposium on Circuits and Systems*. 51–54.

J. C. Lo, S. Thanawastien, T. R.N. Rao, and M. Nicolaidis. 1992. An SFS Berger check prediction ALU and its application to self-checking processor designs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 11, 4, 525–540.

S. Mitra and E. J. McCluskey. 2000. Which concurrent error detection scheme to choose? In *Proceedings of the International Test Conference.* 985–994.

J. Monteiro, S Devadas, P. Ashar, and A. Mauskar. 1996. Scheduling techniques to enable power management. In *Proceedings of the 33rd IEEE/ACM Design Automation Conference* (*DAC'96*). 349–352.

T. Mudge. 2001. Power: A first-class architectural design constraint. *Computer* 34, 4, 52–58.

C. Nagendra, M.J. Irwin, and R.M. Owens. 1996. Area-time-power tradeoffs in parallel adders. *IEEE Trans. Circuits Syst. Express Briefs* 43, 10, 689–702.

A. Namazi, Y. Sedaghat, S.G. Miremadi, and A. Ejlali. 2009. A low-cost fault-tolerant technique for Carry Look-Ahead adder. In *Proceedings of the 15th IEEE International On-Line Testing Symposium (IOLTS'09)*. 217–222.

B. Parhami. 2000. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press.

S. Peng and R. Manohar. 2005. Fault tolerant asynchronous adder through dynamic self-reconfiguration. In *Proceedings of the IEEE International Conference on Computer Design (ICCD'05)*. IEEE, 171–179.

O. A. Pfänder, R. Nopper, H.-J. Pfleiderer, S. Zhou, and A. Bermak. 2008. Comparison of reconfigurable structures for flexible word-length multiplication. *Adv Radio Sci.* 6, 113–118.

S. Purohit, S.R. Chalamalasetti, and M. Margala. 2010. Low overhead soft error detection and correction scheme for reconfigurable pipelined data paths. In *Proceedings of the NASA / ESA Conference on Adaptive Hardware and Systems.* 59–65.

J. M. Rabaey and M. Pedram. 1996. *Low Power Design Methodologies*. Kluwer Academic Publishers.

R. M. Seepers, C. Strydis, and G. N. Gaydadjiev. 2012. Architecture-level fault-tolerance for biomedical implants. In *Proceedings of the International Conference on Embedded Computer Systems*. 104–112.

S. Segars. 1997. ARM7TDMI power consumption. *IEEE Micro* 17, 4, 12–19.

Frederick F. Sellers, Muyue Xiao, and Leroy W. Bearnson. 1968. *Error Detecting Logic for Digital Computers*. McGraw-Hill.

A. Shrivastava, D. Kannan, S. Bhardwaj, and S. Vrudhula. 2010. Reducing functional unit power consumption and its variation using leakage sensors. *IEEE Trans. VLSI Syst.* 18, 6, 988–997.

C. W. Slayman. 2005. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Trans. Device Mater. Reliab.* 5, 3, 397–404.

C. Strydis. 2011. Universal processor architecture for biomedical implants: The SiMS Project. Ph.D. dissertation, Delft University of Technology, Delft, The Netherlands.

Christos Strydis, Georgi N. Gaydadjiev, and Stamatis Vassiliadis. 2006. A new digital architecture for reliable, ultra-low-power systems. In *Proceedings of the 17th Annual Workshop on Circuits, Systems and Signal Processing*. 350–355.

V. Tiwari, S. Malik, and P. Ashar. 1998. Guarded evaluation: pushing power management to logic synthesis/design. *IEEE Trans. Comput-aided Des. Integr. Circuits Syst.* 17, 10, 1051–1060.

W. J. Townsend, J. A. Abraham, and E. E. Swartzlander. 2003. Quadruple Time redundancy adders. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.* 250–256.

M. Vratonjic, B. R. Zeydel, and V. G. Oklobdzija. 2005. Low- and ultra low-power arithmetic units: design and comparison. in *Proceedings of the IEEE International Conference on Computer Design*. 249–252.

Received February 2013; revised May 2013; accepted June 2013