Design of a Low-Energy Data Processing Architecture for WSN Nodes

Cedric Walravens and Wim Dehaene
K.U.Leuven ESAT-MICAS, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
Email: {cwalrave,wdehaene}@esat.kuleuven.be

***Abstract*—Wireless sensor nodes require low-energy components given their limited energy supply from batteries or scavenging. Currently, they are designed around off-the-shelf low-power microcontrollers for on-the-node processing. However, by employing more appropriate hardware, the energy consumption can be significantly reduced. This paper identifies that many WSN applications employ algorithms which can be solved by using parallel prefix-sums. Therefore, an alternative architecture is proposed to calculated them energy-efficiently. It consists of several parallel processing elements (PEs) structured as a folded tree. Profiling SystemC models of the design with ActivaSC helps to improve data-locality. Measurements of the fabricated chip confirm an improvement of 10–20x in terms of energy as compared with traditional MCUs found in sensor nodes.**

I. Introduction

Wireless Sensor Network (WSN) nodes essentially consist of sensors, a radio and a micro-controller (MCU) combined with a limited power supply, e.g. battery or energy scavenging. Since radio transmissions are very expensive in terms of energy, they must be kept to a minimum in order to extend node lifetime [12]. As a result, data communication must be traded for more on-the-node computation. In addition, the variety of WSN applications –ranging from medical monitoring, to environmental sensing, industrial inspection and military surveillance– demands a significant degree of flexibility. This is the main reason why sensor nodes are still designed around off-the-shelf low-power MCUs like the Atmel Xmega [1], STM8L [14] and TI-MSP430 [15]. To improve energy-efficiency, this paper introduces an alternative low-power ASIC approach for WSN data processing in sensor nodes without sacrificing too much of the flexibility found in traditional MCUs. First, Section II identifies that many WSN applications use algorithms which can be solved by employing parallel prefix-sums. Then, Section III presents the proposed architecture for solving parallel prefix-sums energy-efficiently. A SystemC-based profiling methodology is used to identify bottlenecks and improve data-locality of the architecture. Finally, Section IV provides measurements of the silicon chip which uses 10–20x less energy compared to related work.

II. Proposed approach

A. WSN applications and on-the-node data aggregation

Notwithstanding the seemingly vast nature of WSN applications, a set of basic building blocks for on-the-node processing can be identified. Common on-the-node operations performed on input data collected directly from the node's sensors or through in-the-network aggregation include filtering, fitting, sorting and searching [11]. We found that these can be expressed as parallel prefix-sums operations. Prefix-sums can be calculated in a number of ways [3], but we chose the binary tree approach [13] because its flow matches the desired on-the-node data aggregation. This can be visualised as a binary tree of Processing Elements (PEs) across which input data flows from the leaves to the root (Fig. 1, left). This topology will form the fixed part of our approach, but in order to serve multiple applications, flexibility is also required. The tree-based data flow will therefore be executed on a datapath of programmable PEs which provides this flexibility.

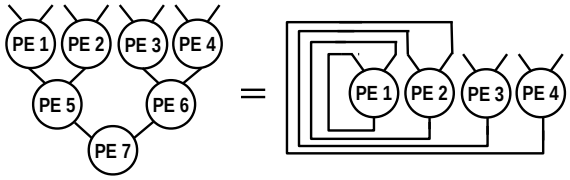

Fig. 1: A binary tree (left, 7 PEs) is functionally equivalent to the novel folded tree topology (right, 4 PEs) used in this architecture.

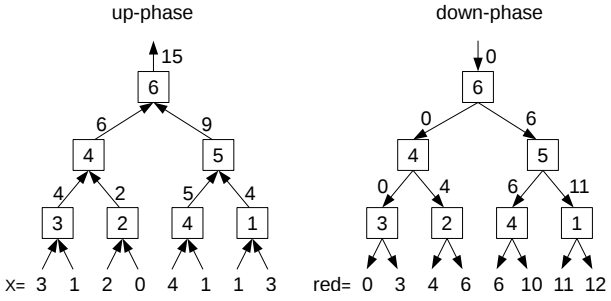

Fig. 2: Finding the reduced-prefix-sums for + on a tree.

B. Parallel prefix-sums

Given a binary closed and associative operator $\oplus$ with identity element I and an ordered set of n elements $[a_0, a_1, a_2, \cdots, a_{n-1}]$, the *reduced-prefix-sums* is the ordered set $[I, a_0, (a_0 \oplus a_1), \cdots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-2})]$, while the *all-prefix-sums* is the ordered set $[a_0, (a_0 \oplus a_1), \cdots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1})]$, of which the last element $(a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1})$ is called the *prefix-sum*.

For example, if $\oplus$ is a simple addition, then the prefix-sum of the ordered set $[3, 1, 2, 0, 4, 1, 1, 3]$ is $\Sigma a_i = 15$. The standard procedure to calculate the prefix-sum on a binary tree (Fig. 2) requires one *up-phase*, starting from the leaves. Each

978-3-9810801-8-6/DATE12/©2012 EDAA

PE along the way up executes exactly one store-and-calculate operation on its inputs. The desired result is found at the root. Getting the reduced-prefix-sums $[0, 3, 4, 6, 6, 10, 11, 12]$ requires an additional *down-phase*, now starting back from the root. Each PE forwards the incoming value to the left (the root uses the identity element) and sends the calculated result to the right. The all-prefix-sums can be found by simply shifting the reduced-prefix-sums to the left and putting the prefix-sum found earlier at the end. Further details can be found in literature [3]. Possible applications that relate to WSNs include peak detection, polynomial evaluation for model-fitting, lexically compare strings, add multi-precision numbers, delete marked elements from arrays and quick sort [4].

C. MCUs and the von Neumann bottleneck

Modern MCUs are still based on the von Neumann architecture [10], or a minor variation thereof. Every algorithm operating on the contents of main memory must send vast numbers of data words back and forth between the CPU and main memory, making it a bottleneck [2] which consumes a significant amount of energy. Also, the lack of task-specific operations found in today's ultra-fast RISC (Reduced Instruction Set Computer) processors leads to inefficient execution, which results in longer algorithms, significant memory book keeping and a worsened energy-performance overall. To avoid the von Neumann bottleneck, data-locality is required. Furthermore, this locality must be realised with a minimum of overhead. Only then energy can be conserved. By exploiting the tree topology for the parallel prefix-sums algorithm, data is automatically pushed through the tree and brought to the PEs, so locality is indeed spontaneously realised.

III. Design Methodology

A. Macro-architecture : towards a folded tree

First, an un-timed (UT) SystemC model of the binary tree of PEs (Fig. 1, left) is written based on the findings of the previous section. ActivaSC [16] is used to profile models. Pipelining is one of the benefits of the binary tree [13]: as soon as a layer of PEs finishes, the results are passed on and calculations can recommence based on the output data from the proceeding layer. However, a straight-forward binary tree implementation costs a significant amount of area as n inputs require $p = n - 1$ PEs. To reduce area and power, pipelining can be traded for throughput. The idea is to *fold* the tree back onto itself to maximally reuse the PEs. In doing so, p becomes proportional to $n/2$ and the area is cut in half. Note that also the interconnect is reduced. On the other hand, throughput decreases by a factor of $log_2(n)$ but since the sample rate of different relevant physical phenomena does not exceed 100 kHz [9], this leaves enough room for the trade-off to be made. This newly proposed folded tree topology is depicted in Fig. 1 on the right.

Moving forward, the model is adjusted to represent the folded tree topology (Fig. 1, right) and processing delays are added. The profiling of this approximately-timed (AT) SystemC model identified a significant amount of unneeded activity. When data moves down the folded version of the tree, the number of nodes which do calculations decreases by a factor of 2 (e.g. for 8 PEs: 8,4,2,1). However, all nodes remain active. This was solved by using handshaking signalling to arrange the data transfer between PEs and keep them in an idle state when no new input data is present.

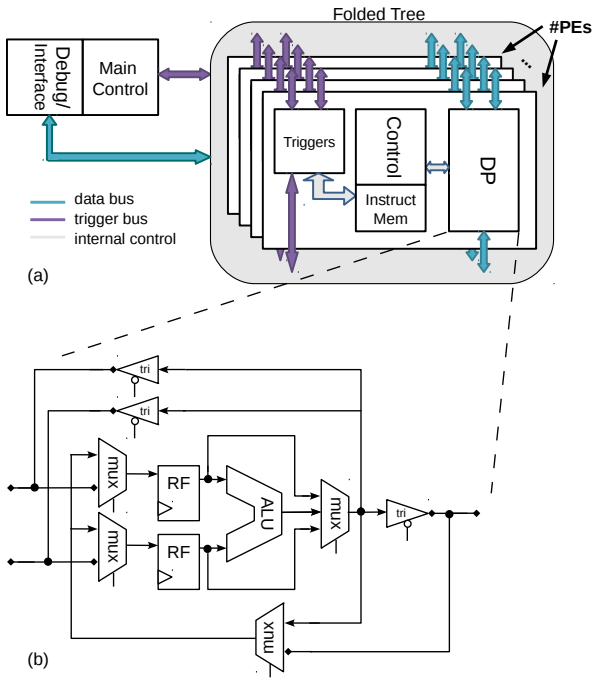

Fig. 3: Schematic overview of the whole design: (a) top-level view with folded tree (4 PEs), (b) detail of one PE data path.

Combining all this into a cycle-accurate model (CA) led to also use the introduced handshaking for steering clock gating signals. ActivaSC can readily provide a comparison of data bus activity with and without handshaking. For a folded tree with 8 PEs, a reduction of at least 60% in terms of activity is realized for the case of one single up-phase. The handshaking overhead is minimal considering the fact that the data bus is much wider (16 bit) than the single bit request-acknowledge handshakes and considering the beneficial impact of keeping unused PEs idle. In addition, this handshaking is also reused to control the bidirectional implementation of the data bus to support both the up- and down-phase. This greatly simplifies the interconnection of the PE structure.

B. Micro-architecture : towards triggered PEs

Fig. 3 gives a detailed overview of the global implementation. The handshaking triggers manage the data flow across the bidirectional data bus throughout the folded tree architecture. They activate the PEs only when new data is available and in such a way that they functionally become a binary tree in up- and down-phase. Each PE within the folded tree is an identical instance of the same generic PE element. It consists of a micromachine which interacts with its own data path. All data path control signals are provided locally. Muxes select external data, stored data or the previous result as the next input for the data path. The data path contains an Algorithmic Logical Unit (ALU) with register files (RF) at the inputs for operand isolation. These RFs become the distributed data

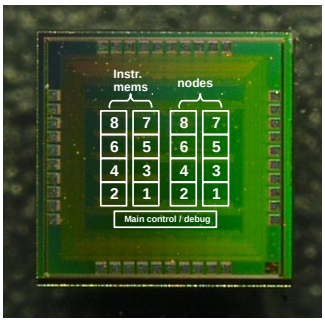

Fig. 4: Die photograph of the implemented processor with 8 PEs.

memory of the whole system. They are the only clocked elements within the data path. As data flows through the tree it is constantly kept local to its designated operation. This is one of the goals of this work, which effectively removes the von Neumann bottleneck and saves power. A PE takes 6 (down-phase) or 7 (up-phase) cycles to process one instruction which can be divided in three stages: (1) *Preparation*, which acknowledges the data and starts the core when input triggers are received (1 cycle); (2) *Execution*, which performs the load-execute-jump stages to do the calculations and fetch the next instruction pointer (4 cycles); and (3) *Transfer*, which forwards the result by triggering the next PE in the folded tree path on a request-acknowledge basis (1-2 cycle). This is tailored towards executing the store-and-calculate operation of the parallel prefix algorithm on a tree as described earlier. Combined with the flexibility to program the PEs using any combination of operators available in their data path, the folded tree has the freedom to run a variety of parallel-prefix applications [4]. Furthermore, the folded tree can implement other sequential (non parallel prefix) algorithms by running it on one single PE. However, this does no longer exploit data-locality and reintroduces a von Neumann bottleneck. In addition, the memory available to such an application is severely limited as only the local RFs are accessible. To alleviate both somewhat, the folded tree structure can also be configured as a chain of PEs which can regain some data-locality and has access to about half of the total distributed memory. Referring to the right-hand side of Fig. 1, this is achieved by starting at the outer right node (PE4) and passing the results onwards while effectively hopping along a chain of roughly half the PEs (PE4, PE2 and PE1).

IV. Experimental validation

The folded tree processor has been fabricated in 130 nm standard cell CMOS (Fig. 4) containing eight 16-bit PEs which can process 16 inputs simultaneously. Each support 14 arithmetic and logical operations with associated flag tests and jump commands. The register files within the PEs can store 4 data words per input. The design targets 20-80 MHz operation at 1.2 V, using VLIW processing of 36-bit instructions.

The chip has separate power domains so the power consumption of separate parts can be measured. Off-the-shelf memory IP macros were originally used to implement the PE's instruction memories. These memory macros are in no way optimised for power. Therefore, the energy consumption of the folded tree PE cores is first measured separately, excluding the instruction memory. The upper table in Fig. 5 gives the dynamic energy and leakage power for one PE core under full stress with varying data inputs. It consumes 42 uW or 2.1 uW/MHz including 0.3 uW leakage. Thanks to the macro-architectural choices (Sect. III-A) the factor 3 difference between the active and idle PE consumption can be fully exploited to reduce power. To make a fair comparison against existing MCUs later on, a power-optimised register-based replacement for the small instruction memories was developed for a follow-up design. It was taken through full P&R, so parasitic extraction of this sign-off layout allowed to accurately simulate its power. These power values are given in last column of the PE table in Fig. 5. To validate the measurements and to check whether the derived values for a single PE are correct, they are combined in an estimate for the folded tree with 8 PEs. When such tree executes an up-phase, it will take 4 steps to reach the root (Sect. II-B). Thanks to the handshaking, the number of active PEs is cut in half as 8,4,2,1 for each step. The number of idle PEs increases accordingly as 0,4,6,7. This makes a total of 15 active PEs and 17 idle PEs. By combining this information with the PE's consumption, the folded tree consumption can be estimated. As can be seen in the lower table of Fig. 5 the estimate closely matches the measured values for the folded tree. The last column also takes the instruction memories into account. Overall, the folded tree processor consumes 257 uW or 13 pJ/cycle including memories.

1 Processing Element	Active PE core	Idle PE core	PE Instr. Mem.
Dynamic energy/instr (pJ)	14.6	4.7	2.10
Leakage power (uW)	0.3	0.3	0.01
Total power @ 20 MHz (uW)	42.0	13.7	6.0

Folded Tree (up-phase)	Estimate	Measured	incl.Instr. Mem.
Total dynamic energy (pJ)	298.8	289.2	356.4
Leakage power (uW)	2.5	2.5	2.6
Total power @ 20 MHz (uW)	215.9	209.1	257.2

Fig. 5: Leakage power and dynamic energy for one PE and the folded tree with 8 PEs executing an up-phase under nominal conditions (20MHz,1.2V).

Core	Arch	DP (bits)	Process (nm)	VDD (V)	Clk (MHz)	E/instr (pJ)	Norm. E/Instr (pJ)
SNAP/LE	RISC GP	16	180	1.8	200	218.0	157.4
BitSNAP	RISC GP	16	180	1.8	54	54.0	17.3
Smart Dust	RISC GP	8	250	1.0	0.5	12.0	18.0
one single PE	CISC GP	16	130	1.2	20	4.2	4.2

Fig. 6: Energy per instruction of related work [5][6][17], normalised to 130nm,1.2V,16bit.

A standard benchmark suite of applications for WSN systems does not exist though some initial attempts have been made [8][11]. Especially regarding academic work, readers are often left with only the energy-per-instruction metric to compare different systems. Fig. 6 presents a summary of related academic work. The given energy/instruction values are normalised to the presented work using following formula:

$$E_{norm} = E_{orig} \times 130nm/L \times (1.2V/V_{dd})^2 \times 16bit/W \quad (1)$$

given energy per instruction E_{orig}, process L, supply V_{dd} and datapath bitwidth D of the other system. This work requires at

Core	Arch	DP (bits)	Memory (KB)	Process (nm)	Supply Current (uA)	VDD (V)	Clk (MHz)	Energy/cycle (pJ)	Normalized E/cyc (pJ)	Algo Unit (cycles)	Algo Unit Norm. E (pJ)
ATxmega128D4	RISC GP	8	8	250	1100	3.0	2	1650	274.6	8	2196
STM8L101	CISC GP	8	1.5	130	900	3.0	8	338	108.0	8	864
TI MSP430F550x	RISC GP	16	4	180	2840	3.0	20	426	49.2	8	394
OpenMSP430	RISC GP	16	2	130	765	1.2	20	45.9	45.9	8	367
This work – 1 PE	CISC GP	16	0.5	130	40	1.2	20	2.4	2.4	7	17
This work – Tree	CISC Tree	16	4	130	214	1.2	20	12.9	12.9	7	90

Fig. 7: Comparing total energy for the algorithmic unit sequence (load-execute-store-jump)

least 4.2x less in terms of energy per instruction. The notion of an instruction, however, might significantly differ especially as WSN systems often employ specific ISAs and specialised hardware to reach extreme energy efficiency. This is the case for this work since the benefit of the parallel prefix-sums framework cannot be fully quantified using the small-scale energy-per-instruction metric.

A better metric for comparison is the energy per algorithmic unit (AU). The AU is a sequence of frequently used steps in the target applications. To calculate this metric, a complete data sheet with full instruction set and detailed power measurements is needed. In contrast to academic work, this information is readily available for many commercial MCUs. Given the context of WSN applications, the AU is defined as a load-execute-store-jump sequence which is predominantly present in data processing algorithms that loop over data arrays. For each MCU, the total number of cycles for the AU sequence is calculated. Each time, the most efficient instructions are chosen from each MCU's specific instruction set. The energy per cycle is based on the information found in the data sheet and normalised using (1). Fig. 7 presents the details of this comparison. The OpenMSP430 [7], which is an open-source model of the widely-used MSP430 MCU, is also included. A single PE outperforms other MCUs by at least 20x in terms of energy, requiring only 2.4 pJ per cycle or 16.8 pJ per AU at equal clock speed. This illustrates that, if memory demands are limited, the single PE or chained PE sequential solution is still more energy efficient.To correctly compare the MCUs with the 8 PEs in the folded tree, the parallel aspect of the latter needs to be taken into account. As derived earlier, a folded tree of 8 PEs will execute 15 AU's over 4 time periods or 3.75 on average. In other words, the folded tree must be compared to the equivalent of 3.75 MCUs. The folded tree then outperforms the closest competitor MSP430 by at least 15x in terms of energy.

Finally, despite the lack of standardised benchmark algorithms, a selection of four relevant algorithms (Sect. II-B) is used to compare the energy requirements against the OpenMSP430. This closest competitor, as indicated by the previous experiments, comes with an accurate sign-off simulation model. It allows to measure the energy consumption of the MSP430 even more accurately than using the AU metric. The results are presented in Fig. 8. The folded tree outperforms the MSP430 by 8–10x in terms of energy and at least 2–3x in terms of execution time. Note that this speed gain can be traded for even more energy-efficient execution, by lowering the supply voltage until an equal throughput is reached. Operating

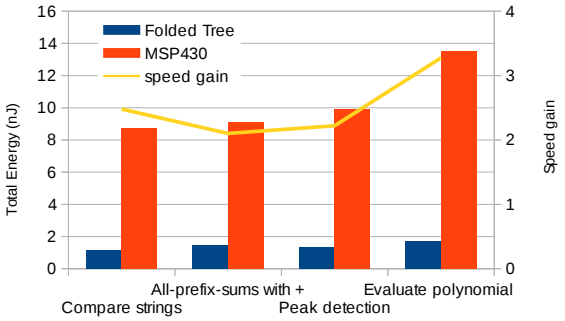

Fig. 8: Total energy consumption of example algorithms (20 MHz,1.2V)

at half the frequency (10 MHz) and a minimal supply voltage of 0.79 V the processor now consumes about half the energy. A single active PE core will only consume 0.95 uW/MHz including leakage. Overall, the folded tree processor now consumes down to 80 uW or 8 pJ/cycle and running the example algorithms it outperforms other MCUs by at least 20x in terms of total energy.

V. Conclusion

This paper identified that the data processing algorithms found in many WSN applications can be solved by using parallel prefix-sums methods. An architecture was developed, in order to calculate these in an energy-efficient manner. Mainly owing to improved data-locality, tree folding and controlled triggering of the PEs, a low-energy execution of down to 8 pJ/cycle or at least a 20x better energy-efficiency is realised.

References

[1] Atmel Corp. Xmega D4 datasheet, 2010.
[2] J. Backus. Can programming be liberated from the von Neumann style? In *Communications of the ACM*, volume 21. ACM, 1978.
[3] G. Blelloch. Scans as primitive parallel operations. *Computers, IEEE Trans.*, 38(11):1526–1538, 1989.
[4] G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-1990, Carnegie Mellon University, Nov 1990.
[5] V. Ekanayake et al. SNAP/LE: An ultra low-power processor for sensor networks. In *ACM SIGOPS OS Rev.*, volume 38. ACM, 2004.
[6] V. Ekanayake et al. BitSNAP. *IEEE Computer Society*, 2005.
[7] O. Girard. OpenMSP430 processor core, available at opencores.org.
[8] M. Hempstead et al. Tinybench. In *IEEE Proc. 29th Local Computer Networks Conf.*, pages 585–586. IEEE Computer Society, 2004.
[9] M. Hempstead et al. Survey of hardware systems for wireless sensor networks. *Journal of Low Power Electronics*, 4(1):11–20, 2008.
[10] J. Hennessy and D. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, ISBN 0-1237-0490-1, 4th edition, 2007.
[11] L. Nazhandali. Sensebench. In *IEEE Proc. Workload Characterization Symposium*, pages 197–203. IEEE, 2005.
[12] V. Raghunathan et al. Energy-aware wireless microsensor networks. *Signal Processing Magazine, IEEE*, 19(2):40–50, 2002.
[13] P. Sanders and J. Träff. Parallel prefix (scan) algorithms for MPI. *Recent Adv. in Parallel VM and MPI*, pages 49–57, 2006.
[14] ST Microelectronics. STM8L101 datasheet, 2010.
[15] Texas Instruments. MSP430F550x datasheet, 2010.
[16] C. Walravens et al. ActivaSC: a highly efficient and non-intrusive extension for activity-based analysis of SystemC models. In *DAC'09*.
[17] B. Warneke et al. An ultra-low energy microcontroller for smart dust wireless sensor networks. In *ISSCC 2004*, pages 316–317. IEEE, 2004.