Seeking low-power synchronous/asynchronous embedded systems: an FIR implementation case study

Ali Skaf[1], Jean Simatic[2] and Laurent Fesquet[2]

[1] Syrian Private University - Damascus - Syria
ali.skaf@spu.edu.sy

[2] TIMA Laboratory INPG/CNRS - Grenoble - France
jean.simatic@imag.fr – laurent.fesquet@imag.fr

Abstract

Seeking low-power consumption high-performance embedded systems has been at the center of interest for researchers around the world for the last decades, especially with the recent boom of different hand-held battery-operated mobile connected devices. The new trends and needs of faster, smarter and smaller internet connected systems, also known as the IoT, require developing very low-power embedded systems including actuators, sensors and signal processors. In this paper, we focus on the architecture optimization efforts to reduce the required activity using the FIR filter as a demonstration example. The new optimized implementation of the FIR filter was compared with other synchronous and asynchronous FIR filter versions realized using the ALPS framework developed at TIMA laboratory. The obtained FIR architecture exhibits around 33% area 52% power consumption reduction compared to the best previous synchronous implementation. We plan to use these results to improve the automatically generated datapath of the high-level synthesis tool of our framework (ALPS-HLS).

Keywords: low-power embedded systems, IoT, optimized synchronous design, event-driven sampling, asynchronous circuits.

1. Introduction

Reducing the power consumption of embedded systems and devices raise critical power issues, especially in battery-powered connected applications or the Internet of Things (IoT). These issues have pushed research communities around the globe to seek smart very low-power design solutions. Many possible tracks have been explored in order to reduce the power consumption of such systems invoking removing, or at least reducing, the clock activity, being the major source of power consumption in digital systems. In asynchronous systems, we look for simply eliminating the clock signal at the expense of new design methodology. While in synchronous systems, we try to reduce the clock activity whenever possible using basically different techniques of clock gating to switch off the inactive parts of the embedded system. Nevertheless, data processing tasks that represent a major part of power consumption cannot be avoided. Thus, there is an obvious need of implementing the data processing tasks using the less activity demanding architecture. In the traditional synchronous design schemes, designers tend to reduce the clock activity of the inactive parts, or to propose schemes based on adapting the data processing with the nature of treated data. For instance, non-uniform sampling can be used digital signal processing systems [1]–[5]. Unlike typical uniform sampling strategies, the sampling takes place only when the signal crosses certain threshold values (statically or dynamically defined), providing natural data compression for communication or storage. The so called level-crossing sampling scheme (LCSS) [6] restricts the switching activity through the circuits, reducing by consequence the system's power consumption.

Other solutions rely on simply removing the clock signal and replace it with handshake protocol giving what is commonly named asynchronous systems. These systems are well-suited to handle the data flow irregularity of non-uniform sampling schemes. Moreover, asynchronous designs consume power only when data are being processed, and they can virtually offer higher performance than worst case, increased robustness, and thus better electro-magnetic compatibility [7]–[9].

In both the cases of synchronous and asynchronous systems, optimization efforts were centered on the system dataflow and how to control the system functioning to reduce the communication with the outside world. Yet there are issues that merit further exploring related to the data processing in the datapath itself and how to better choose the processing blocs and how to (re)organize the datapath in order to reduce the power consumption. This effort can even imply using other arithmetic representation systems and tools like, for instance, redundant number systems as the Signed Binary Digit (SBD) system and other on-line arithmetic operators [10] [11].

This work proposes to show the impact of such an optimization effort applied to Finite Impulse Response (FIR) filter implementation. FIR filters are widely used in the digital signal processing blocs of communicating embedded systems and devices. Previous implementations carried out at TIMA laboratory demonstrated the outcome of applying different power consumption reducing methodologies as the non-uniform sampling, the synchronous/asynchronous manual and automatic implementations using the dedicated framework ALPS (Architectural tools for ultra-Low

Power (event-driven) Systems) and the synchronous-dedicated HLS tool AUGH [12] .

We start by reviewing the basics of FIR algorithm in section I. Section II will summarize the previous implementations carried out at TIMA and their characteristics. The proposed architecture optimization is detailed in section III, while the filter obtained synchronous implementation performance will be given in section IV along with the comparison with the other versions before concluding.

I- FIR basics

To filter an input signal x generating an output filtered signal y, the FIR algorithm can be expressed as follows: If we denote x_i, y_i and h_i as respectively the i^{th} samples of the input signal x, output signal y, and impulse response. Then, the output samples are computed applying the equation:

$$y_k = \sum_{i=0}^{N} x_{k-i} \cdot h_i$$

The straightforward parallel implementation is composed of (N+1) registers to form a delay line, (N+1) multipliers to compute the different ($x_j.h_i$) products, and N adders to form y_k. To reduce the required hardware, we can choose a serial implementation using only one multiplier and one adder with an extra storage register for the partial results and a dedicated ROM (or decoder) to supply the h_i values. This design requires also a control part to generate the needed datapath control signals. In the case of a 32-order filter with samples represented on 8 bits, the obtained serial architecture is shown in Fig. 1.

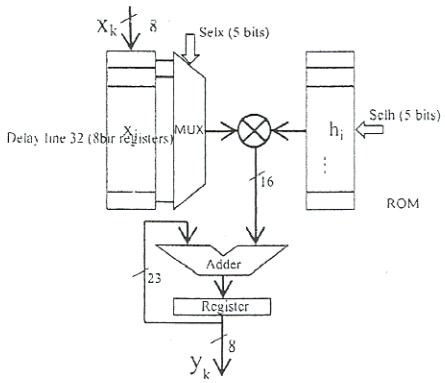

Fig.1 : 32-order FIR filter serial implementation.

It is to be noticed that:
1- In case of a traditional synchronous functioning, for each entered input sample, 32 clocks are required to compute the corresponding output. The commercial synthesis tools, such as used among others in Altera Quartus II or Xilinx Vivado, can simply generate the datapath and control parts starting from an HDL description.
2- This architecture can be desynchronized using already existing asynchronous High-Level Synthesis (HLS) tools like TiDE [13] and Balsa [14] but the outcome depends on the designer ability to write appropriate HDL code.

II- Previous FIR synchronous/asynchronous implementations

Efforts have been deployed to build a complete framework to help designers to develop asynchronous designs. The methodology consists of using a synchronous tool AUGH [15], which provides support for a standard programming language – a subset of ANSI C – and a wider set of possible transformations such as loop unrolling and memory type selection. AUGH generates a synchronous synthesizable RTL description of the datapath and the specification of the synchronous FSM. Then, the synchronous FSM is desynchronized to derive the asynchronous control. The asynchronous FSM uses distributed late-capture controllers [16]. The late-capture protocol requires dedicated conditional branching components, which were specified and designed [17]. The proposed desynchronization method is specific to FSMs, unlike generic methods such as [18].

To validate the effectiveness of the framework, three FIR filter implementations, of a 32-order low-pass filter with a normalized sampling frequency of 1 and a cut-off normalized frequency of 1/44.1, were discussed and compared in terms of area, computation time and energy consumption [17], namely:
1) a typical synchronous FIR filter (Fig.1),
2) an asynchronous filter for non-uniformly sampled signals manually designed at the RTL level,
3) the same asynchronous filter automatically generated by ALPS HLS.

In fact, in implementations 2) and 3) the case of non-uniform sampling was considered using an interpolation-based algorithm [19]. This approach is based on setting discrete levels and then sampling the input signal whenever it crosses one of the two neighboring levels. The interpolation scheme adopted in these implementations is a piece-wise constant interpolation, yet other schemes are possible but lead to more complex computations [20]. The hardware required for the datapath is more complex than implementation 1) as two multipliers are needed. Notice that these asynchronous implementations have actually a synchronous-like datapath operated by an asynchrounous control part.

The comparison of the three different implementations, in 350nm and in 40nm technologies, in response to a sum of sines (SS) and an ECG signals was carried out. Taking the synchronous implementation as a reference, automated asynchronous design 3) gave the smallest area. For both test signals, the asynchronous filters consume less energy than the synchronous filter.
For SS, the asynchronous filters require 4 to 6 times less energy. For the signal ECG, the energy consumption is divided by 23 to 43, due to the sparsity of the signal well suited to non-uniform sampling. For the 40 nm technology, the energy consumption benefit is improved by 20%. Indeed, the 40 nm library standard cells include Muller gates – which simplify the asynchronous control part. The manual asynchronous filter consumes 23% to 35% less energy than the filter generated from the proposed high-level synthesis method.

In the following sections, we propose a new architecture to implement the FIR filter datapath that can be used in level-crossing or non-uniform sampling case.

III- New optimized FIR implementation

Another track to investigate in the quest of reducing the power consumption is to work at the data processing and computing blocs. In fact, lower consumption can be achieved using the less activity demanding architecture. Using the LCSS allows us to reduce the power consumption at the system level getting rid of the need for a precise ADC with a digital output represented on a large number of bits. We can even use a set of simple comparators instead. At the FIR filter level, the LCSS enables us to reduce the datapath width and thus the activity of the computation blocs.

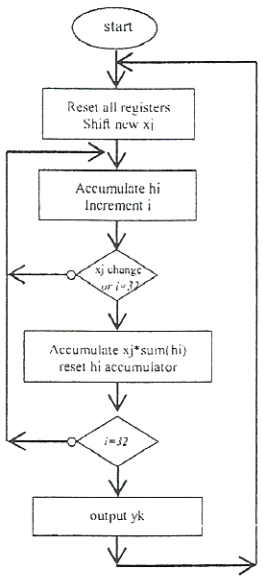

Fig.2 : Proposed FIR computation algorithm.

For the FIR filter, we propose an optimized architecture making benefit of the LCSS and modifying the data processing execution by invoking the multiplication once in a while. In this case, everything happens as if we had the same sample value x_k for a certain number of theoretical sampling periods. So we can reduce the number of multiplications. The equation for the FIR filter becomes:

$$y_k = \sum_j x_j \cdot \sum_i h_i$$

Where x_j is the current input sample to be multiplied by the sum of h_i values till the next level crossing takes place. The computation algorithm is as follows (Fig.2):

At the end of computation we have the sum of all (i , j) = N+1. Notice that in case the input signal stays for long time (more than $(N+1)/f_s$) without crossing a new level, an output is to be generated and a new computation cycle is initiated. The obtained architecture is presented in Fig. 3. The selection command is the current counter value i. The compare bloc generates a signal *Next* that indicates the value change of the current treated input sample selected at the delay line. The accumulated h_i value has to be multiplied by the current value (stored at the register after the multiplexer). Then the h_i accumulator has to be reset to zero to compute the new sum of h_i corresponding to the next x_j value.

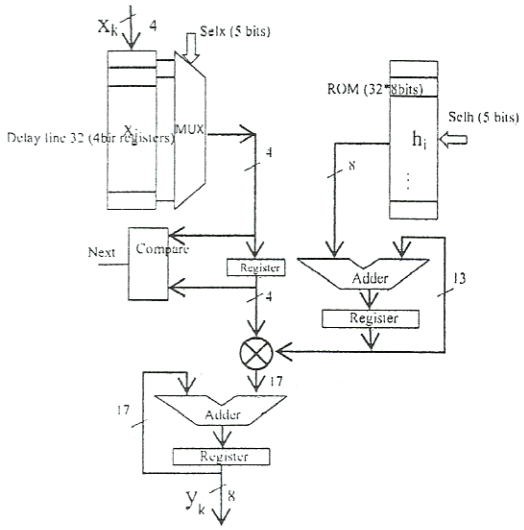

Fig.3 : Optimized FIR filter architecture.

Here we took the same features of previous non-uniform sampling implementations in terms of the number of levels (16). The h_i accumulator has to be 13-bit wide to cover the worst case corresponding to the accumulation of all stored 32 values of h_i. The final accumulator must have 17 bits to cover all the possible values.
We developed the VHDL code for the datapath and the control part implemented an FSM. This architecture is robust and provides correct functioning even if the signal crosses a neighboring level each sampling period.

All the registers receive the system clock which is at least 32 times faster than the sampling frequency. We opted for using the enabled-clock registers rather than the gated-clock; in order to guarantee the code portability when targeting an FPGA implementation in which special clock distribution resources are foreseen to provide the best performance.

IV- Performance comparison

After validating the design, the fully synchronous synthesized version was compared with the previous three FIR implementations, in both 350nm and 40nm CMOS technologies.

The table of Fig. 4 summarized the measured values for the new version compared with the previous three implementations for the two target technologies. The measurements were done for the cases of a sum of sines (SS) and an ECG input signals. We clearly see that the optimized architecture implementation occupies the smallest area of all cases with a reduction of 33% compared to the typical synchronous version. As for the required energy, we notice that the optimized synchronous version is situated between the synchronous and asynchronous versions. It allows up to 52%

reduction compared with the typical synchronous implementation.

#	FIR Implementation	Area (μ^2) for Technology		Energy Consumption (μ joules)			
				350 nm Technology		40 nm Technology	
		350 nm	40 nm	SS	ECG	SS	ECG
1	Synchronous	171000	2070	94.72	94.72	0.653	0.653
2	Manual Asynchronous	161000	1900	18.9	2.814	0.099	0.015
3	Automatic Asynchronous	146000	1830	28.44	4.07	0.158	0.023
4	Optimized Synchronous	114000	1560	52.96	46.21	0.511	0.404

Fig.4 : Different FIR filter implementation results.

The asynchronous versions figures indicate that it would be interesting to desynchronize the current optimized architecture to virtually reach the lowest power consumption.

Fig.5 : Output signals for the FIR filter implementations.

As far as the quality of filtering is concerned, Fig. 5 shows the output signals corresponding to the studied input SS and ECG signals. The presented optimized synchronous implementation seems to give the best performance.

Conclusions

We have shown in this paper the importance of well choosing the data processing blocs in the datapath to reduce the overall power consumption, which is especially important for the battery-operated embedded systems and IoT applications. Optimization efforts to reduce the required computation activity using the FIR filter were presented as a demonstration example. The new optimized implementation of the FIR filter was compared with other synchronous and asynchronous FIR filter versions developed at TIMA laboratory. The obtained FIR architecture exhibits around 33% area 52% power consumption reduction compared to the best previous synchronous implementation. Further power consumption improvement is expected be obtained via desynchronizing the proposed optimized FIR filter architecture.

References

[1] E. Allier, G. Sicard, L. Fesquet, and M. Renaudin, "Asynchronous level crossing analog to digital converters," Measurement, vol. 37, no. 4, pp. 296 – 309, 2005, 8th Workshop on ADC Modelling and Testing.

[2] F. Akopyan, R. Manohar, and A. B. Apsel, "A level-crossing flash asynchronous analog-to-digital converter," in 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2006, pp. 11 pp.–22.

[3] R. L. Grimaldi, S. Rodriguez, and A. Rusu, "A 10-bit 5kHz level-crossing ADC," in 20th European Conference on Circuit Theory and Design (ECCTD), 2011, pp. 564–567.

[4] P. Martínez-Nuevo, S. Patil, and Y. Tsividis, "Derivative level-crossing sampling," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 62, no. 1, pp. 11–15, 2015.

[5] B. Bidégaray-Fesquet and L. Fesquet, "Levels, peaks, slopes. . . which sampling for which purpose?" in 2nd International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP), 2016.

[6] J. Mark and T. Todd, "A nonuniform sampling approach to data compression," IEEE Transactions on Communications, vol. 29, no. 1, pp. 24–32, 1981.

[7] K.-L. Chang, J. Chang, B.-H. Gwee, and K.-S. Chong, "Synchronous logic and asynchronous-logic 8051 microcontroller cores for realizing the Internet of Things: A comparative study on dynamic voltage scaling and variation effects," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 3, no. 1, pp. 23–34, 2013.

[8] N. Paver, P. Day, C. Farnsworth, D. Jackson, W. Lien, and J. Liu, "A low-power, low noise, configurable self-timed DSP," in 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1998, pp. 32–42.

[9] K. van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters, "Asynchronous circuits for low power: a DCC error corrector," Design Test of Computers, IEEE, vol. 11, no. 2, pp. 22–32, 1994.

[10] A. Skaf and A. Guyot: "VLSI Design of On-line Add/Multiply Algorithms" International Conference on Computer Design (ICCD 93), Cambridge, USA, October 1993.

[11] A. Skaf and A. Guyot: "SAGA: the First On-line Arithmetic Coprocessor" - VLSI 95- New Delhi- January 1995.

[12] A. Prost-Boucle, O. Muller, and F. Rousseau, "Fast and standalone design space exploration for high-level synthesis under resource constraints," Journal of Systems Architecture, vol. 60, no. 1, pp. 79–93, 2014.

[13] S. Nielsen, J. Sparso, J. Jensen, and J. Nielsen, "A behavioral synthesis frontend to the Haste/TiDE design flow," in 15th IEEE Symposium on Asynchronous Circuits and Systems (ASYNC), 2009, pp. 185–194.

[14] A. Bardsley, "Balsa: An asynchronous circuit synthesis system," Master's thesis, University of Manchester, 1998.

[15] A. Prost-Boucle, "AUGH: Autonomous and user guided high-level synthesis," Software, http://tima.imag.fr/sls/research-projects/augh/.

[16] J. Simatic, A. Cherkaoui, R. P. Bastos, and L. Fesquet, "New asynchronous protocols for enhancing area and throughput in bundled data pipelines," in 29th Symposium on Integrated Circuits and Systems Design (SBCCI), Aug 2016.

[17] J. Simatic, P. Alexandre, A. Cherkaoui, R. P. Bastos, and L. Fesquet "A High-Level Synthesis Tool for Designing Ultra-Low Power Asynchronous Systems" Submitted to IEEE Transactions on Computer Design Conference - 2016

[18] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 10, pp. 1904–1921, Oct 2006.

[19] F. Aeschlimann, E. Allier, L. Fesquet, and M. Renaudin, "Asynchronous FIR filters: towards a new digital processing chain," in 10th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), April 2004, pp. 198–206.

[20] L. Fesquet and B. Bidégaray-Fesquet, "SPASS 2.0: Signal processing for asynchronous systems," Software, http://ljk.imag.fr/membres/Brigitte.Bidegaray/SPASS/ [accessed 2014-01-28], May 2010