Implementation of autocorrelation algorithm in VHDL for UVP instrumentation

Isadora Fernanda Zappe Schmidt[1], Fabio Rizental Coutinho[1], Andre Luis Stakowian[2], Cesar Yutaka Ofuchi[2], Marco Jose da Silva[3], Flavio Neves Jr[2], and Rigoberto Eleazar Melgarejo Morales[4]

[1] Dep. of Electronics Engineering, Federal Univ. of Technology - Paraná (UTFPR), R. Cristo Rei 19, Vila Becker, Toledo, Parana, 85902-490, Brazil
[2] Graduate School of Electrical Engineering and Computer Science (CPGEI), Federal University. of Technology - Paraná (UTFPR), Av. 7 de Setembro 3165, Curitiba, 80230-901, Parana, Brazil
[3] Institute of Measurement Technology, Johannes Kepler University Linz, Altenberger Str. 69, 4040, Linz, Austria
[4] Mechanical & Materials Engineering Postgraduate Program (PPGEM), Federal University of Technology - Paraná (UTFPR), Av. 7 de Setembro 3165, 80230-901, Curitiba, Parana, Brazil

Ultrasound velocity profiler (UVP) instrument can be applied in a variety of applications from industrial (oil, food processing, etc.), to environmental (hydrology, sewer, etc.) to energy (nuclear, hydropower plants, etc.). UVP equipment measures flow velocity using an autocorrelation method or phase-shift method. This method is established on the phase estimation for sequential ultrasonic pulses of a complex demodulated signal. The present work aims to implement in VHSIC Hardware Description Language (VHDL) a velocity estimation algorithm based on the autocorrelation method. First, the method was implemented in MATLAB® to be used as a reference. Then, the algorithm was implemented in VHDL in a MAX10 Field Programmable Gate-Array (FPGA) using the DE10-Lite Board from Terasic. The VHDL implementation process digitalized data using IEEE 754 standard floating-point number representation with single precision. Validation was performed using previous data acquired from a one-phase horizontal pipe flow. The algorithm implemented in VHDL presented a relative estimated velocity error below 0.00003%, consuming 4,623 total logic elements and 28 embedded multiplier elements from MAX10 FPGA.

Keywords: Ultrasound, Doppler effect, Flow measurement, Field programmable gate array, Phase-shift estimation

1. Introduction

Ultrasound Velocity Profiler (UVP) is a well-established method for liquid velocity measurements. It consists of measuring the Doppler effect observed in the echoes returned by a pulsed ultrasound wave [1]. The next step in the evolution of this technique is the use of multiple transducers for two-dimension velocity measurements or transducer arrays for velocity imaging applications. A field-programmable gate array (FPGA) is an integrated circuit capable of computing a high density of data with low latency. It is also very efficient for performing parallel processing of several channels. And it is also flexible enough to be applied at lower or higher computational demand applications. The phase shift or the use of an autocorrelator for estimating the Doppler velocity is the main computational demand of a UVP instrument. This work aims to describe the implementation of an autocorrelation method in VHSIC Hardware Description Language (VHDL) for use in a MAX10 Field Programmable Gate-Array (FPGA).

1.1 Background

The work of [2] presented an implementation of a pulsed wave Doppler ultrasound in a Virtex-5 FPGA. They estimated the velocity using a 128-point discrete Fourier transform implemented in Verilog language. It was reported that a total of 1,159 slice resources and 3,223 slice registers were required from the FPGA for the proposed technique.

Another similar study is from [3], which was based on the Cyclone III FPGA family (Altera-Intel, San Jose, CA, USA). They implemented a Doppler frequency estimator that uses the peak and centroid of the power spectral density to compute velocity. It requires 10,308 logic elements for FFT computation running with a 105 MHz of clock frequency.

2. Methodology

2.1 Autocorrelation method theory

The autocorrelation algorithm is based on computing the phase for a set of ultrasound-pulsed emissions. The phase shift, φ, of the ultrasound echoes are directly related to the velocity component in the transducer axis direction by [4]

$$v_1 = \frac{c\varphi}{4\pi f T}, \tag{1}$$

where c denotes the sound velocity in the considered medium, f is the transducer central frequency and T is the period between emissions. The phase, φ, is estimated as the argument of the autocorrelation function, as [5, 6]

$$\varphi = tan^{-1}\frac{Im[R(T)]}{Re[R(T)]}. \quad (2)$$

The input of the autocorrelation function is the IQ-demodulated ultrasound signal or the signal complex envelope [7]. Thus, the autocorrelator algorithm must process complex numbers, $r = x(n) + iy(n)$, where n denote the ultrasound emission number. The autocorrelation function for a set $N-1$ ultrasound pairs of emissions is obtained by

$$R(T) = \frac{1}{N-1}\sum_{n=0}^{N-2} r^*(n)r(n+1)\,, \quad (3)$$

where r^* denotes the complex conjugate of r.

2.2 Experimental set-up

VHDL algorithm was implemented in the DE-10 Lite board from Terasic. This hardware provides a large capacity MAX10 FPGA from Altera, which has up to 50.000 logic elements. It also features a 64MB SDRAM, an Arduino UNO R3 expansion connector, analog-to-digital converter, VGA output, etc. The system setup is shown in Fig.1. Digital data are provided to DE-10 Lite kit through a SPI interface. An Arduino UNO R3 board acts as a bridge between a PC and the FPGA board. In the SPI context, the Arduino is configured as the bus master and the FPGA as the slave. The MATLAB ® Support Package for Arduino ® Hardware was used to allow MATLAB to interactively communicate with the Arduino board. Previously acquired data from a real flow is loaded on Matlab environment which sends it to the DE-10 Lite board through an Arduino kit. The FPGA stores the data received in the SDRAM and then evaluates the autocorrelation function. In Matlab, the same computation performed in the FPGA is also done. The results obtained from FPGA are returned to the PC through SPI. Both results, from Matlab and FPGA, are compared to assess computation accuracy.

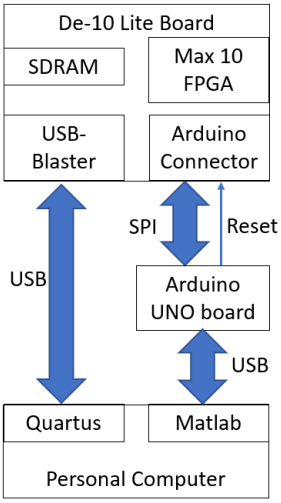


Figure 1: System set-up.

VHDL code is developed using Quartus Prime Lite Edition (version 18.1, from Intel). The code compiled is implemented at MAX10 FPGA through a USB connection. Since Eq. (1) comprises a multiply and accumulate computation, it was chosen to use a floating-point data representation to accommodate a large range of real values. At the FPGA, digital data was represented using 32 bits using IEEE 754 standard for single precision. The VHDL implementation of floating-point multiply and accumulate was based on [8]. In Matlab, it was chosen to use a 64 bits (or double precision) data representation since it will be used as a reference.

2.3 Autocorrelation implementation

The autocorrelation function was implemented in VHDL language as depicted in Fig. 2. Data is received from Arduino by the SPI controller. Arduino UNO was set up as an SPI master with a clock polarity and phase at high level. The interface was configured to transfer 16 bits of data at each transfer with a 20 MHz clock. Data from SPI controller is buffered and sent to be stored in SDRAM through the Write SDRAM module (Fig. 2). When a total of N ultrasound emissions are stored, the autocorrelator begins to read the first ultrasound emission sample then applies Eq. 3. This calculation is repeated for the next emission until it reaches the $N-1$ emission. The result from the autocorrelator is sent from SPI to Matlab(PC) for comparison purposes.

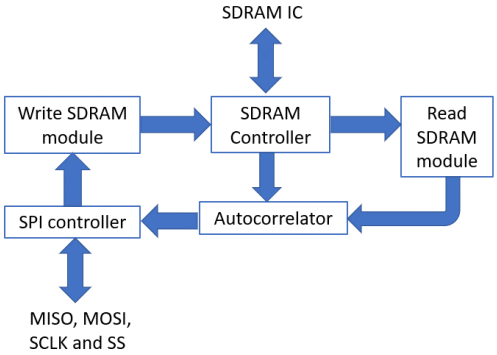


Figure 2: VHDL software block diagram.

Initialization, read and write low-level operations at SDRAM are performed by the SDRAM controller. This module used an open-source code from [9]. The SPI controller was responsible for the SPI communication. It also concatenates each pair of 16 bits of data to form 32 bits single precision word. The Write SDRAM module is responsible for buffering and issuing a write command for the SDRAM. The data for the autocorrelator is delivered by the Read SDRAM module which control the requests of readings and buffering.

The autocorrelator block consist of a state machine that control the order of execution and two VHDL components: a floating-point multiplication (fp_mul) and a floating-point summation (fp_add) component (Fig. 3). These components are instantiating 4 times in VHDL to implement the Eq. 3 as depicted in Fig. 3. The simplified

autocorrelator state machine (Fig. 4) comprises of 5 states. In the first state, the autocorrelator is waiting for a start signal which is issued by SDRAM controller after all the data are correctly stored. At state 1, the next sample, $r(n+1)$, is read at SDRAM. At the first execution $r^*(n)$, Rp and Ip are loaded with zero values. At state 2, the process depicted in Fig. 3 is executed. At state 3 it is checked if the $N-1$ ultrasound pairs of emissions were reached. If not, the content of R_SUM and I_SUM are shifted to Rp and Ip, respectively, and the content of $r(n+1)$ is shifted to $r(n)$. Reaching the end of emissions, the results of R_SUM and I_SUM are sent to the SPI controller.

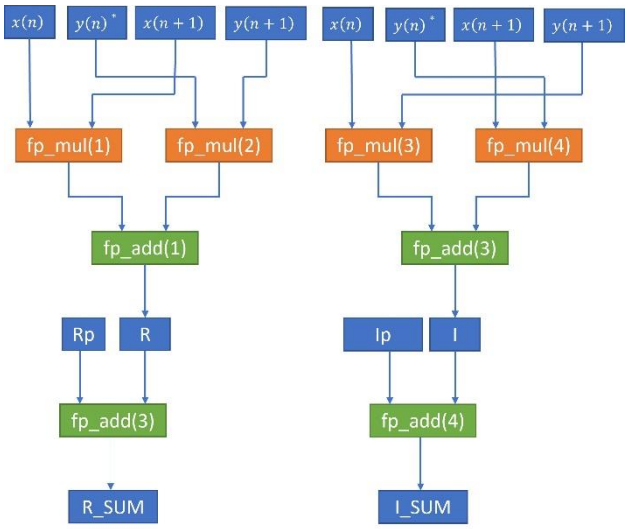


Figure 3: Multiply and accumulate component diagram.

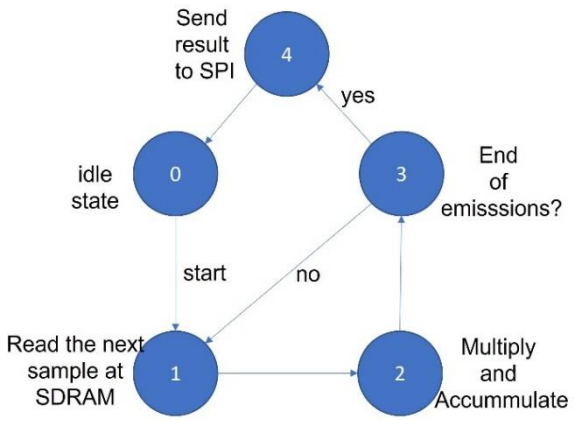


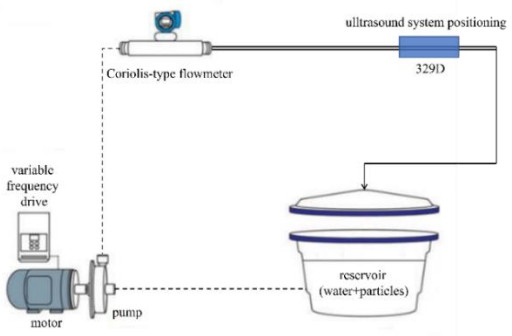
Figure 4: Autocorrelator state diagram.

2.4 Flow apparatus

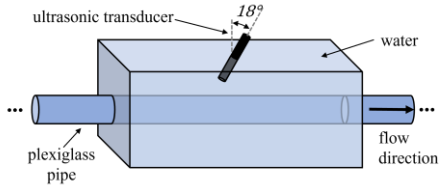
Input data, used for evaluating the FPGA autocorrelator, were obtained through a pipe flow experiment as shown in Fig. 5. In the experiment, the centrifugal pump – driven through an inverter of variable frequency – circulates the liquid in an acrylic tube whose inner diameter is of 25.9 mm (D). A trace powder with a 1.07 g/cm3 density was added to the reservoir at a concentration of 4 g/L. Water flow is measured by a Coriolis-type flow meter. A 4 MHz ultrasound transducer (Met-flow S.A, TX4-5-8) was positioned at 329D from the pipe entrance. It was coupled with a box filled with water and it was angled by 18º to the pipe normal (Fig. 5b).

Data acquisition was performed with a PXI System from National Instruments, model NI5752R. This system can acquire data at 50 Msamples/s and store data for offline processing. A pulse repetition frequency of 6,005.99 Hz was set-up. A mean flow velocity of 1.0 m/s was adjusted in the flow for the data collection.

Velocity processing is done offline using Matlab environment. Each velocity was measured using 128 emissions Data acquired from each emission was IQ demodulated [7]. After demodulation, the digitized data was filtered by a matched filter centered in 4 MHz and 4-cycles pulse. Finally, it was converted to a double precision (64 bits) at Matlab environment to be used as a reference. For FPGA processing data precision were reduce to 32 bits (single precision).



(a)



(b)

Figure 5: (a) Flow apparatus. (b) Ultrasound system positioning

To assess the accuracy of the proposed method, three range gates or depths {0.027D, 0.054D, 0.5D}, with D=25.9 mm, were chosen as shown in Fig. 6. The FPGA processor only calculates the autocorrelation value, thus velocity estimated by the FPGA is evaluated at Matlab using the autocorrelation values computed by the FPGA, using Eq. (1) and Eq. (2).

3. Results

The results shown in Tables 1 and 2 were the autocorrelation evaluation and velocity obtained with the data from the FPGA processor. The relative error between the estimates from FPGA and Matlab was very small, below 0,00003%. Thus, indicating that the proposed method is feasible to be used in terms of accuracy. This small relative error shown in Table 1 is because the autocorrelation performed by MATLAB® considers

double precision number representation (64 bits) whereas the FPGA algorithm uses single precision representation (32 bits). Velocity data estimated also show a very good agreement, (Table 2).

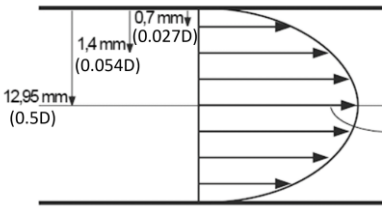


Figure 6: Range gates depths analyzed.

In terms of FPGA resources, a total of 4,623 logic elements were used (Table 3). Considering only the autocorrelation implementation, 4,123 logic elements and 352 registers were required (Table 3). Since MAX10 FPGA has 49,760 logical elements, the implementation used approximately 9.3% of the total logic elements of the integrated circuit. Considering the embedded multiplier elements, MAX10 has 288 9-bit multiplier elements. Thus, approximately 10% of the embedded multiplier (Table 3) was used in the proposed method. Compared with the results from [3], this implementation can estimate velocity with less than half of the FPGA resources. To compute the autocorrelation function, the FPGA spent 15 clock cycles for each emission. This is equivalent to spending 0.3 µs for each emission (for a 50 MHz clock). Considering the experimental setup condition used (128 emissions), the computation of the autocorrelation function will spend 38.4 µs.

Table 1: Accuracy results of the VHDL autocorrelator.

Depth (mm)	Relative Error - real part (%)	Relative Error - imaginary part (%)
0.7	$+1.633\times10^{-5}$	-1.265×10^{-5}
1.4	-2.419×10^{-5}	-1.498×10^{-6}
12.95	$+2.514\times10^{-5}$	5.746×10^{-6}

Table 2: Accuracy results from velocity.

Depth (mm)	Velocity - Matlab (m/s)	Velocity - FPGA (m/s)	Relative error (%)
0.7	0.52074182	0.52074176	-1.152×10^{-5}
1.4	0.78348715	0.78348714	-1.276×10^{-6}
12.95	1.08836469	1.08836460	-8.269×10^{-6}

Table 3: FPGA resources used.

Entity	Total Logic Elements	Total Registers	Embedded Multiplier 9-bit elements
Autocorrelator	4,123	352	28
SDRAM	417	267	0
SPI controller	83	97	0
Total	4,623	716	28

4. Summary

In this work, an autocorrelation method for the measurement of fluid flow velocity was implemented in VHDL for FPGA processing. The data processed was represented using a floating point (IEEE 754 standard) with single precision (32 bits). The proposed technique presented a very small relative error regarding autocorrelation or velocities estimation. Due to this, as future work, we recommend assessing the accuracy of a half-precision floating-point data representation (16 bits). Using fewer bits for data representation will imply using less FPGA resources and optimized hardware.

The implementation proposed used approximately 9.3% resources of the MAX10 FPGA. This result shows that this hardware may be capable of processing up to 10 ultrasound transducers simultaneously.

4. Acknowledgments

This work was carried out with the support of the following Brazilian agencies: Fundação Araucária de Apoio ao Desenvolvimento Científico e Tecnológico do Paraná (grant 026/2020) and CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico (grant 406881/2021-9 – Chamada CNPq/MCTI/FNDCT Nº18/2021, Faixa A - Grupos Emergentes). The authors are grateful for their support.

References

[1] Takeda Y: Ultrasonic Doppler fluid flow, Springer, (2012).
[2] Page A and Mohsenin T: An efficient & reconfigurable FPGA and ASIC implementation of a spectral Doppler ultrasound imaging system, IEEE 24th Int. Conf. on Application-Specific Sys., Arch. and Processors, 42(4), (2013), 198-202.
[3] Ricci S and Meacci V: FPGA-Based Doppler Frequency Estimator for Real-Time Velocimetry, Electronics, 9(456), (2020).
[4] Ofuchi C Y *et al.*: Extended Autocorrelation Velocity Estimator Applied to Fluid Engineering, ISUD 9, (2014),109-112.
[5] Kasai C, *et al.*: Real-time two-dimensional blood flow imaging using an autocorrelation technique, IEEE Trans. on Sonics and Ultras., SU-32(3), (1985),458-464.
[6] Loupas T, *et al.*: An axial velocity estimator for ultrasound blood flow imaging, based on a full evaluation of the Doppler equation using a two-dimensional autocorrelation approach, IEEE Trans. on Ultr., Fer., and Freq. Ctrl, 42(4), (1995), 672-688.
[7] Haykin S: Communications Systems, John Wiley & Sons Inc, New York (2001).
[8] Deschamps J P, *et al.* Guide do FPGA implementation of arithmetic functions, Springer, Dordrecht, 2012.
[9] Placha, A Sdram controller, open-source code, 2019. https://github.com/Arkowski24/sdram-controller.