Speed-Optimized Implementation of Fast Chirplet Decomposition Algorithm on FPGA-SoC

Austin Fite, Mikhail Gromov, Tianyang Fang and Jafar Saniie

ECASP Research Laboratory (http://ecasp.ece.iit.edu)
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago, IL, U.S.A.

***Abstract*— In ultrasonic nondestructive evaluation (NDE) of materials an essential step in characterizing an ultrasonic signal is decomposing the patterns of multiple interfering echoes. The Chirplet Transform (CT) is a powerful method to analyze the echoes in an ultrasonic signal. However, CT analysis is computationally heavy and impractical. Motivated by achieving real-time execution of the CT this research presents a speed-optimized implementation of the chirplet functions on FPGA. Chirplet echo generation used in Fast Chirplet Decomposition (FCD) Algorithm for ultrasonic signal analysis necessitates the frequent generation of chirplet functions with a 6-degree of freedom associated with chirplet parameters including the amplitude scaler; the time of arrival; the Gaussian envelope scaler; the phase of the chirplet; the center frequency and the frequency sweep. By minimizing the processing time of the chirplet generation, the FCD algorithm can be implemented efficiently on FPGA System-on-Chip (SoC). This study presents the hardware realization of the chirplet function on FPGA which is 37 times faster compared to using a Teensy 4.0 microcontroller, and 146 times faster than a highly popular Raspberry Pi 4.0 single board computer.**

Keywords—FPGA, Signal Processing, Ultrasonic, Optimization.

I. Introduction

In the NDE of materials using ultrasound, determining echo patterns in an ultrasonic signal provides useful information about any deformities and multipath interference [1-3]. This information is important for ultrasonic medical imaging and industrial nondestructive testing. The chirplet signal decomposition algorithm is an effective method for echo characterization of a channel and provides useful information about the ultrasonic propagation path through materials [1]. Motivated by this problem, this research presents speed-optimized hardware solutions for Fast Chirplet Decomposition (FCD) intended for use on an FPGA System-on-Chip. The foundation for exploring FCD algorithm improvement is presented in [2]. This algorithm demands frequent generation of chirplet functions with 6 degrees of freedom.

To achieve better execution time for chirplet generation, our research provides a scheme to implement the multiplications and LUT in hardware. The hardware multiplications and additions will use floating-point arithmetic for a large portion of the calculations using principles from [4] and [5]. A major motivation for using floating points is to take advantage of the large dynamic range that the floating point format provides while maintaining 24 significant bits [6]. Additionally, multiple multiplications and LUTs will be performed in parallel on the FPGA. For the chirplet generation, multiple chirplet generators will be undersampled and run in parallel to generate a data bus capable of generating multiple chirplet samples per clock cycle. Using undersampled parallel chirplet generators is a design choice inspired by [6] and [7], which provides information on how to increase the sample rate of data conversion when working with limitations in the data converters themselves. Although this research does not directly use data converters in the FPGA design, the undersampled parallel architecture is similar to what this paper describes.

In Section II the Fast Chirplet Decomposition (FCD) algorithm is introduced, which will describe the process by which a signal may be decomposed to constituent chirplets. Section III describes the chirplet transform and details of the different transform parameters. Section IV provides an FPGA implementation to accelerate a key component of the algorithm, chirplet generation. Section V will evaluate the performance of the FPGA implementation against a software-based approach using a Teensy 4.0 microcontroller and Raspberry Pi 4.0 single-board computer, and characterize the improvements that are made by using this hardware accelerator. Section VI will conclude the paper by explaining how the hardware accelerator can be used for the FCD algorithm and how it may be used for other applications.

II. Fast Chirplet Decomposition

The chirplet decomposition algorithm decomposes a signal containing multiple overlapping chirplets into its constituent parts. Each chirplet may be represented by a set of six parameters to be used for later analysis of an ultrasonic signal channel (see Equations (1) and (2)). Fig. 1 shows the flowchart for the fast chirplet decomposition algorithm. The algorithm steps are:

1. Find the maximum location of $s(t)$ in the time domain and use it as the initial guess of time-of-arrival and the starting point of iteration.
2. Estimate the center frequency, which maximizes the chirplet transform, given the initial guess of time-of-arrival.
3. Estimate the time-of-arrival, which maximizes the chirplet transform, given the estimated center frequency from the previous step.

4. Estimate the center frequency, which maximizes the chirplet transform, given the new estimated time-of-arrival from Step 3.

5. Check convergence: If $\Delta\tau < \tau$ and $\Delta f < flim$ (here, τlim and $flim$ are predefined convergence conditions), then go to Step 6; otherwise, go to Step 3.

6. Estimate the amplitude β and the remaining parameters $\alpha 2$, ϕ, and $\alpha 1$ successively.

7. Obtain the residual signal by subtracting the estimated echo from the signal.

8. Calculate the energy of the residual signal (Er) and check convergence ($Emin$ is predefined convergence condition): If $Er < Emin$, the algorithm is completed; otherwise, go to Step 1.

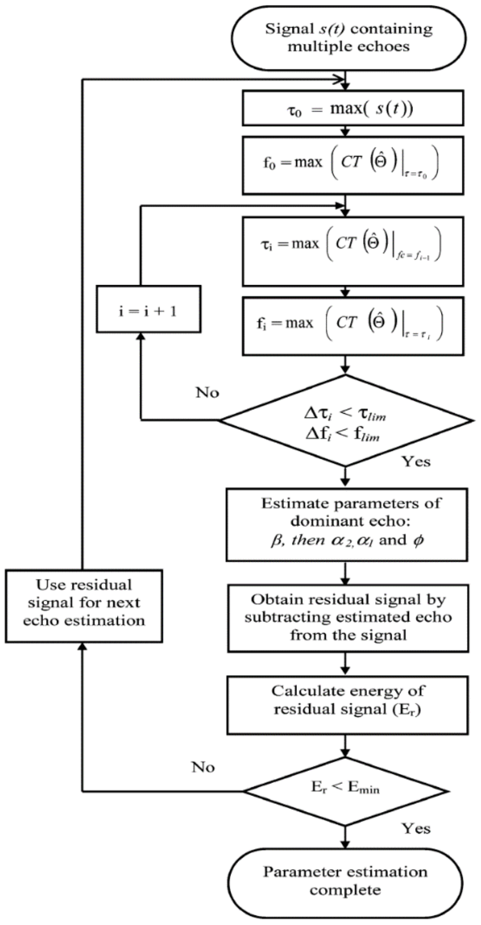
Signal $s(t)$ containing multiple echoes
$\tau_0 = \max(s(t))$
$f_0 = \max\left(CT(\hat{\Theta})\big|_{\tau=\tau_0}\right)$
$\tau_i = \max\left(CT(\hat{\Theta})\big|_{fc=f_{i-1}}\right)$
$f_i = \max\left(CT(\hat{\Theta})\big|_{\tau=\tau_i}\right)$
i = i + 1
No
$\Delta\tau_i < \tau_{lim}$ $\Delta f_i < f_{lim}$
Yes
Estimate parameters of dominant echo: β, then α_2, α_1 and ϕ
Use residual signal for next echo estimation
Obtain residual signal by subtracting estimated echo from the signal
Calculate energy of residual signal (E_r)
No
$E_r < E_{min}$
Yes
Parameter estimation complete

Fig. 1. Fast Chirplet Decomposition Flowchart [2].

In each step besides step 1, where a parameter estimation is made, the chirplet transform is invoked with specific parameter estimates and a measured reference signal. The chirplet transform is a computationally complex problem performed iteratively throughout the algorithm. The number of chirplets transformed needed to complete the algorithm depends on the resolution of the parameter sweep, the number of embedded chirplets in the input signal, and the amount of acceptable residual energy after chirplet decomposition. It is common to perform a very large number of chirplet transforms throughout this algorithm. Therefore, if the chirplet generation may be accelerated via a digital hardware implementation, significant progress can be made in reducing the amount of time needed to execute the algorithm. This paper provides an FPGA module that may be used with the FCD algorithm as a means of hardware acceleration to reduce the time needed to execute the FCD algorithm.

III. Chirplet Transform Optimization

To find the appropriate chirplet parameters to best model a reference signal, the chirplet transform is used to iteratively find the best-fitting parameters [1, 2]. The chirplet transform of a given estimated parameter set $\hat{\theta}$ is defined [8]:

$$CT(\hat{\theta}) = \int_{-\infty}^{\infty} f_\theta(t)\Psi_{\hat{\theta}}^*(t)\,dt \tag{1}$$

The chirplet transform takes two functions as its input: $f_\theta(t)$ and $\Psi_{\hat{\theta}}^*(t)$. Both of these functions are chirplets whose form follows:

$$C(t) = \beta e^{\alpha_1(t-\tau)^2 + j2\pi[\phi + f_c(t-\tau) + \alpha_2(t-\tau)^2]} \tag{2}$$

As shown in Equation (2), there are six main components to a chirplet. β is the amplitude scaler; τ is the time of arrival; $\alpha 1$ is the Gaussian envelope scaler; ϕ is the phase of the chirplet; fc is the center frequency and $\alpha 2$ is the frequency sweep parameter.

For this research, $f_\theta(t)$ represents a measured signal and will be referred to as the reference signal. And $\Psi_{\hat{\theta}}^*(t)$ represents the estimated signal generated from the estimated parameters. In [2] the maximum location of the chirplet transform is used to facilitate a peak search algorithm; the current maximum value is compared against the previous maximum value, and this difference is what determines if the estimated parameter is within a sufficiently appropriate range. This research found that the same effect may be reproduced by cross-correlating the two function inputs to the chirplet transform and only analyzing the center point of the cross-correlation between these two functions instead of a full cross-correlation followed by finding the maximum value of that cross-correlation as defined in [2]. So long as the initial time of arrival estimate is reasonable, the series of center cross-correlation points created by iterative parameter search is still conducive to a peak search algorithm. Then, the chirplet transform is divided into two separate tasks: estimated chirplet generation and a center point cross-correlation. In the following sections, the two tasks and their computation complexity are discussed.

A. Chirplet Generation Estimation

The chirplet generator is used to estimate a chirplet whose output is fed into a cross-correlator module. A chirplet is a signal containing the properties of both a chirp and a wavelet. A chirp is a single-component signal with a varying frequency. A wavelet is a time-limited signal and is generated by multiplying a window function with a signal in the time domain. Fig. 2 shows a chirplet example, note that the window of choice for this chirplet transform is a Gaussian envelope (see Equation (2)).

Chirplet generation is an expensive task due to its multiple exponential functions, both Gaussian and sinusoidal. Calculating these exponentials in software either requires using math libraries or look-up tables for sinusoidal and exponential approximations. Even when using look-up tables in software, each portion of the chirplet equation is estimated individually and cannot be made parallel like with the FPGA implementation. And because the size of the chirplet is usually large at least 512 samples, the calculation of this array becomes time-consuming. Although the calculation of the chirplet signal is O(n), the large size of the array and the exponential functions that comprise the chirplet signal makes it a logical target for optimization.

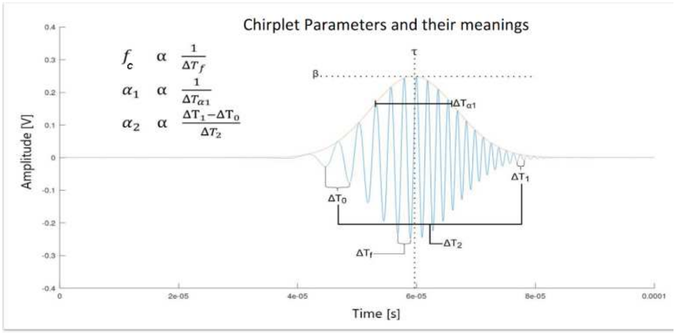

Fig. 2. Chirplet Parameters Related to the Shape of Chirplet Signal.

IV. FPGA Implementation

The FPGA implementation of the FCD is shown in Fig. 3. The top-level algorithm is executed on the FPGA Programmable System (PS), which uses software running on an ARM core, and the Chirplet Transform is implemented on the Programmable Logic (PL), the traditional FPGA logic gate array that hardware description languages target.

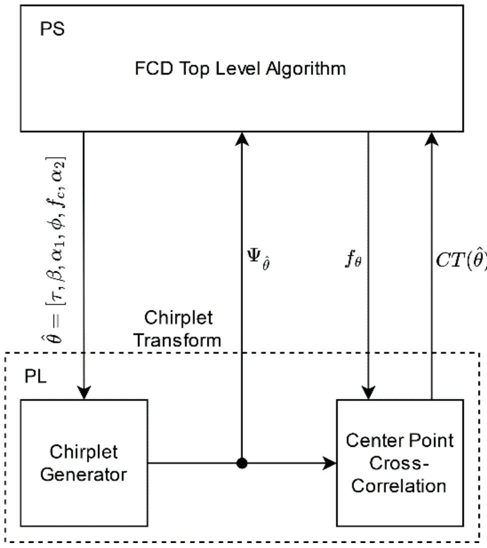

Fig. 3. System-on-Chip Design Diagram for the FCD Algorithm.

The exponential portion of Equation (2) may be broken into subsections which will be generated in parallel to improve the generation speed. First, Equation (2) is broken into two sections, one for the Gaussian envelope and one for the chirp generation:

$$\Psi(t) = \beta\, \Psi_1(t)\Psi_2(t) \tag{3}$$

$$\Psi_1(t) = exp(\alpha_1(t - \tau)^2) \tag{4}$$

$$\begin{aligned}\Psi_2(t) &= exp\big(2\pi j(\phi + f_c(t - \tau) + \alpha_2(t - \tau)^2)\big) \\ &= cos\big(2\pi(\phi + f_c(t-\tau) + \alpha_2(t - \tau)^2)\big) \\ &\quad + jsin\big(2\pi(\phi + f_c(t-\tau) + \alpha_2(t - \tau)^2)\big)\end{aligned} \tag{5}$$

In each case, for $\Psi_1(t)$ and $\Psi_2(t)$, the argument for the exponential functions is generated before using a LUT to approximate the exponential function. For $\Psi_2(t)$, the exponential function is split into real and imaginary components using Euler's identity, and a sine LUT is used to approximate the output.

For a chirplet generator that outputs one sample per clock cycle, the chirplet output sample is represented as a complex-valued number, with 16 bits used to represent the real component and 16 bits used to represent the imaginary component. In addition to the six defining parameters of a chirplet, there is an additional parameter called the time step, which is equivalent to the sample period. This parameter is a constant value and is not one of the parameters that go through parameter estimation. The input parameters to the chirplet generator are represented by 32-bit floating point values. This allows the PS to interface with the chirplet generator intuitively without needing additional parameter scaling to fit a fixed-point format.

Because the input parameters use a floating-point format, the math used in the chirplet generator uses floating-point addition and floating-point multiplication. While the floating-point format allows for more seamless integration of the chirplet generator between the PS and PL, there is a limitation to how many samples the chirplet generator may output before the floating-point resolution becomes inadequate and the output signal becomes corrupted by quantization noise.

To optimize the number of clock cycles needed for parameter estimation, multiple chirplet generators are instantiated in parallel to generate a bus of signals which represent a single chirplet signal, as shown in Fig. 4. Each chirplet generator that is instantiated produces an undersampled chirplet below the necessary Nyquist sampling rate required to reconstruct the chirplet signal. By having each generator delay its output, the resulting output is a reconstructed chirplet where multiple samples are generated in a single clock cycle. Fig. 5 shows an example of how parallel generators work together to generate a signal. Each parallel generator sample is represented by a different marker color. Note that for an individual marker color, the sample rate is too low to reconstruct the original signal properly, while using time-delayed samples in parallel, all of the necessary information is produced to reconstruct the original signal.

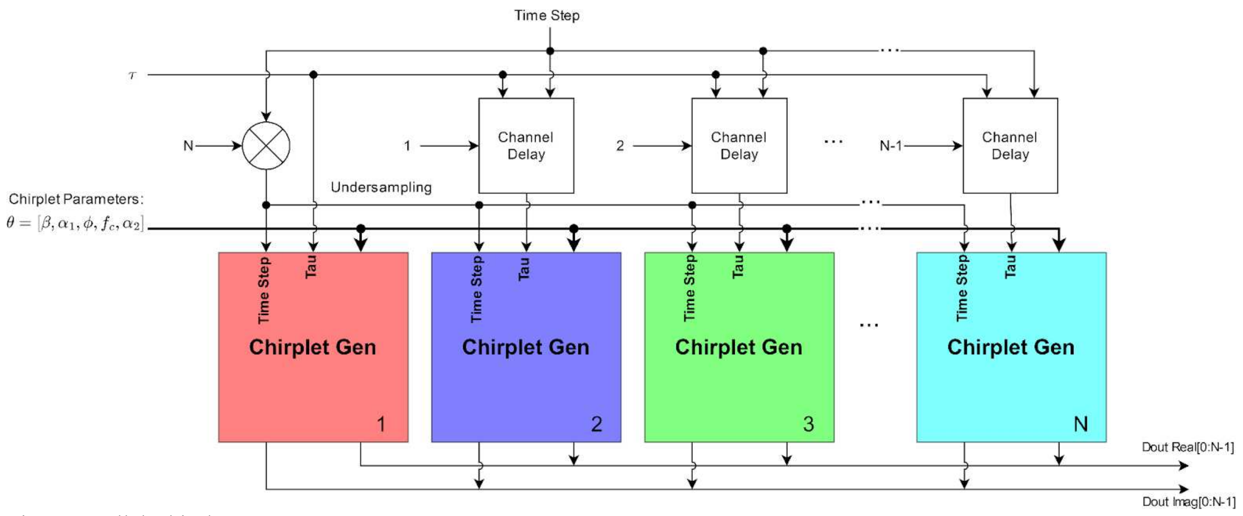

Fig. 4. Parallel Chirplet Generators

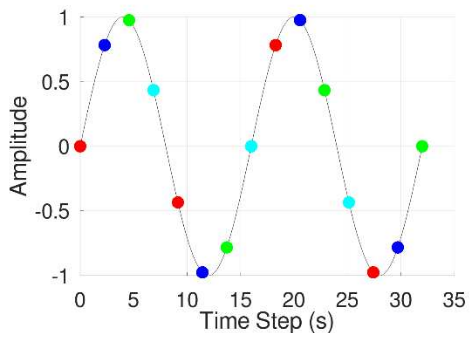

Fig. 5. Undersampled Parallel Output Example

V. Evaluations

To evaluate the FPGA implementation, the two components of the chirplet transform are evaluated: chirplet generation and cross-correlation. Because each chirplet generator is pipelined to achieve a throughput of one sample per clock cycle, the limiting factor to the generation speed becomes a combination of the chirplet generator latency and the number of parallel generators. The FPGA used to evaluate the parallel chirplet generator is the "Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC". LUT lengths of 65536 are used, and eight parallel generators are instantiated. A simulation of a chirplet with 512 samples is carried out. The implemented design uses a 187.5MHz clock, which is the clock frequency used in this simulation to show accurate timing statistics. The parallel generator generates 512 samples in 415.74 ns or 78 clock cycles in the simulation. Simulations also show that the latency between the valid input pulse and the first valid output is 14 clock cycles. This is due to the longest delay path required in an individual chirplet generator to close timing at 187.5MHz. Therefore, increasing the number of parallel generators has diminishing returns due to the fixed latency between the valid input pulse and the first output valid. This implementation was designed to push the chirplet generator to its limit and therefore closes timing with a worst negative slack of 0.008ns, which leaves very little margin for error if the FPGA speed is unknown or known to be slow. For other implementations, it will be necessary to use a slower clock speed to account for an FPGA that comes off the manufacturing line with slower characteristics. Table I shows the utilization report for the parallel chirplet generator. Due to the large FPGA size used, the LUTs, LUTRAMs, FFs, and DSPs used are all in tolerable ranges, allowing for many other modules to be included in the FPGA design along with the chirplet generator. The most used resources are BRAMs, this is due to the large LUTs used in the generators, using smaller LUTs will significantly reduce the BRAM used at the cost of the accuracy of the chirplet generator. Both the timing analysis and implementation report were generated using the full SoC architecture provided for the Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC. Therefore, these numbers are accurate to what a full system would experience rather than the isolated chirplet generator.

Table I. Resource Utilization Report for the Parallel Chirplet Generator

Resource	Available	Utilization	Utilization %
LUT	274080	38079	13.89
LUTRAM	144000	179	0.12
FF	548160	17210	3.14
BRAM	912	612	67.11
DSP	2520	160	6.35
BUFG	404	5	1.24

A large size of 65536 is used for the LUT lengths to minimize the estimation error from using LUTs as an approximation to a function. To evaluate this output, a simulation output is compared to a Matlab output. The parameter values for the chirplet in this simulation are shown in Table II.

Fig. 6 shows the result of the parallel chirplet generator simulation. The top subplot shows both the simulation output and Matlab output. Note that they are essentially indistinguishable from each other when only using qualitative inspection. The bottom subplot shows the calculated error between the FPGA simulation and the Matlab output, the bottom subplot demonstrates that for this parameter set, the maximum error of $6*10^{-5}$ is recorded. This minuscule error is low enough to only contribute a negligible amount of noise power to the chirplet output, thus making this implementation of the parallel chirplet generator adequate for cross-correlation against measured signals without causing false correlation peaks or burying true peaks in noise.

TABLE II. PARAMETER VALUES FOR THE CHIRPLET GENERATION

Parameters	Values
Sample Rate	100 MHz
Time of arrival τ	$2.56\text{x}10^{-6}$ s
Bandwidth Factor $\alpha1$	10^{12}
Center Frequency f_c	5 MHz
Chirplet Rate $\alpha2$	10^{12}
Phase Shift ϕ	0.75 rads
Amplitude β	0.25

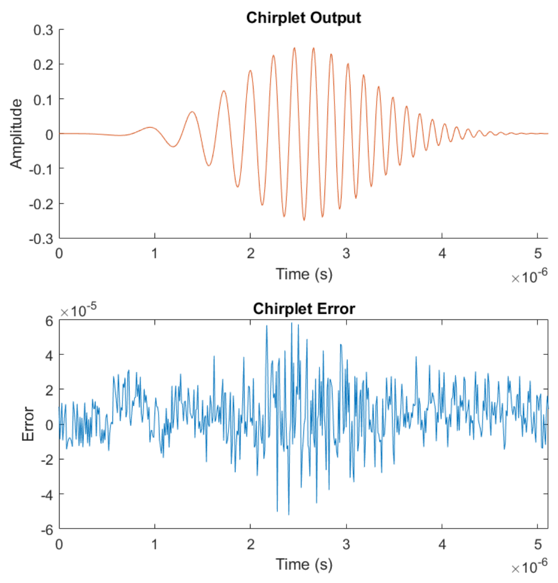

Fig. 6. A Single Chirplet generated from the Parallel Chirplet Generator

To compare the FPGA hardware implementation against a software approach, the chirplet generator code is written in C code for a Teensy 4.0 microcontroller running on a 600MHz clock, and for a Raspberry Pi 4.0 running on a 1500MHz clock. The reason for selecting these devices is to provide a speed comparison against popular embedded DSP software processors. Sine, cosine, and exponential functions are approximated via software LUTs with 2048 values, and all multiplications are done as integer multiplications (as opposed to floating points). The smaller LUT and integer multiplication is done to provide a best-case scenario for the software approach when calculating chirplet samples. The calculation time itself is measured by toggling a GPIO high, calculating 512 samples, and then toggling the pin low. In addition, the switching time of the pin itself is accounted for and subtracted from the final time to provide an accurate measurement of the processing time needed to output 512 chirplet samples.

The final results are shown in Table III. The calculation time for the Teensy 4.0 and Raspberry Pi 4.0 is 15.41us and 60.8us, respectively. Compared to the 415.74ns needed to generate the same information, the parallel chirplet generator improves the calculation time by 37 times compared to the Teensy 4.0 and 146 times compared to the Raspberry Pi 4.0.

TABLE III. DEVICE SPEED COMPARISON FOR CHIRPLET GENERATION

Device	Time to Calculate Chirplet (us)
FPGA Implementation	0.416
Teensy 4.0	15.4
Raspberry Pi 4.0	60.8

This metric is specific to the FPGA implementation used for the “Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC”, tighter pipelining to allow for increased clock speed, reduction in LUT size to increase the clock speed, and increasing the number of parallel chirplet generators are all options that improve the calculation time taken by the FPGA hardware implementation. The process of generating a chirplet is vital for generating a reference to cross-correlate against and is performed many times in the FCD algorithm, hence why it is important to use hardware acceleration to generate this chirplet as quickly as possible.

While the cross-correlation cost function is not the bottleneck for this system, keeping its performance in mind is important. This part of the implementation shares the same FPGA, and was evaluated using a 187.5 MHz clock. Using a length of 512 samples, similar to the one used for the chirplet generation, this module completes a center-point cross-correlation in 23 clock cycles, or 122.667 ns, after the signal is loaded. Both the timing analysis and implementation report was generated using the full SoC architecture provided for the Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC, and are included in Table IV. Similar to the chirplet generator, the LUTs, FFs, BRAMs, DSPs, and BUFGs used are all in tolerable ranges (see Table IV). The most used resources are DSPs, due to a large number of parallel processing blocks.

TABLE IV. RESOURCE UTILIZATION REPORT FOR CROSS-CORRELATION MODULE

Resource	Available	Utilization	Utilization %
LUT	274080	11562	4.22%
FF	548160	8198	1.50%
BRAM	912	28.5	3.13%
DSP	2520	256	10.16%
BUFG	404	4	0.99%

To compare the FPGA hardware implementation against a software approach, the chirplet generator code is written in C code for a Teensy 4.0 microcontroller running on a 600MHz clock, and for a Raspberry Pi 4.0 running on a 1500MHz clock with 512 samples. The final results are shown in Table V. The calculation times for the Teensy 4.0 and Raspberry Pi 4.0 are 6.84 us and 32.41us, respectively. Compared to the 122.7ns needed to generate the same information, the cross-correlation module improves the calculation time by a factor of 56 compared to the Teensy 4.0, and a factor of 264 compared to the Raspberry Pi 4.0.

TABLE V. DEVICE SPEED COMPARISON FOR CROSS-CORRELATION MODULE

Device	Time to Calculate Cross-Correlation (us)
FPGA Implementation	0.1227
Teensy 4.0	6.84
Raspberry Pi 4.0	32.41

VI. CONCLUSION

This paper has provided an architecture for hardware acceleration of chirplet generation intended for use with the fast chirplet decomposition (FCD) algorithm. The point of generating a chirplet is to use the generated chirplet as a signal to correlate against to characterize the component chirps within a measured signal. By using this hardware-accelerated architecture, chirplet data may be sent back to software within the system on chip for cross-correlation, or the chirplet data may be sent to another hardware block for correlation. In either case, a large parallel data bus may be used to communicate between the chirplet generator module and any other module (software or hardware) that it will interface with. Using a large data bus allows for data to be transferred quickly and lets the user take advantage of the parallel nature of the hardware acceleration. When interfacing back to the software, Xilinx supports AXI DMA bus widths of up to 1024 bits, more than the required amount for this hardware implementation. If desirable, this design may be modified to increase the number of parallel generators to take full advantage of the high DMA bus width.

Although this module has been designed as a hardware acceleration tool for the FCD algorithm, other uses for chirplets may find this module (or sections of this module) useful. For example, the LoRa radio communication protocol has proven that a chirp spread spectrum is useful for wireless communication [9,10]. The chirplet generator presented in this paper may be used to generate chirps for the transmission of the chirp spread spectrum, or it may be implemented in the receiver of a chirp spread-spectrum device. Because there is potential to generate chirplets so quickly with this module, it would be possible for the receiver to correlate against several different chirplets in parallel as a type of rake receiver intended to be ready to receive from several unique chirp spread sequences.

REFERENCES

[1] Y. Lu, R. Demirli, G. Cardoso and J. Saniie, "A successive parameter estimation algorithm for chirplet signal decomposition," in IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 53, no. 11, pp. 2121-2131, November 2006, doi: 10.1109/TUFFC.2006.152.

[2] Y. Lu, E. Oruklu and J. Saniie, "Fast Chirplet Transform With FPGA-Based Implementation," in IEEE Signal Processing Letters, vol. 15, pp. 577-580, 2008, doi: 10.1109/LSP.2008.2001816.

[3] P. M. Djuric and S. M. Kay, "Parameter estimation of chirp signals," in IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 38, no. 12, pp. 2118-2126, Dec. 1990, doi: 10.1109/29.61538.

[4] Kralev, J. (2019). Design of floating-point arithmetic unit for FPGA with Simulink®. IEEE EUROCON 2019 -18th International Conference on Smart Technologies. https://doi.org/10.1109/eurocon.2019.8861860

[5] IEEE, "IEEE Standard 754-1985 for Binary Floating-Point Arithmetic". IEEE, 1987. Reprinted in SIGPLAN 22, 2, 9-25.

[6] Mixed-Signal and DSP Design Techniques, Engineering Staff of Analog Devices Inc., printed by Analog Devices, 2000.

[7] Analog Devices. (2001, March 1). *Multiply your sampling rate with time-interleaved data converters*. Multiply Your Sampling Rate with Time-Interleaved Data Converters. Retrieved March 21, 2023, from https://www-dev.cldnet.analog.com/cn/technical-articles/multiply-your-sampling-rate-with-timeinterleaved-data-converters.html

[8] Mann, Steve & Haykin, Simon. (2002). The Chirplet Transform: A Generalization of Gabor's Logon Transform. Vision Interface.

[9] P. D. Prasetyo Adi, Y. Wahyu and A. Kitagawa, "Analyzes of Chirps Spread Spectrum of ES920LR LoRa 920 MHz," 2022 11th Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS), Malang, Indonesia, 2022, pp. 139-144, doi: 10.1109/EECCIS54468.2022.9902922.

[10] A. A. Kherani and K. M. P. Maurya, "Improved Packet Detection in LoRa-like Chirp Spread Spectrum Systems," 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Goa, India, 2019, pp. 1-4, doi: 10.1109/ANTS47819.2019.9118076.