Fixed-Point Processing of the SAR Back Projection Algorithm on FPGA

Don Lahiru Nirmal Hettiarachchi, *Student Member, IEEE,* Eric J. Balster, *Senior Member, IEEE*

***Abstract*—Time-domain back projection (BP) is a widely known method used in Synthetic Aperture Radar (SAR) image formation. Despite its advantages over other image formation algorithms, the BP method is hindered due to its computational complexity and its requirement of higher number of operations and processing power. Recently, Field Programmable Gate Array (FPGA) devices have been used for BP acceleration mainly due to their parallel processing capabilities, reconfigurability, scalability, and low power requirement. This paper presents a new Fixed-point based BP (FxBP) design for FPGA devices and a Floating-point based BP (FlBP) design to compare performance. Both designs are developed with N-Dimensional Range (NDR) structure and Single Work Item (SWI) structure using OpenCL. The FPGA performance is evaluated using a FPGA performance metric (FPM). It is shown that FxBP-NDR and FxBP-SWI designs generate high quality back projected images compared to FlBP designs, while saving 16.87% and 42.54% on logic resources and gaining 17.90% and 91.62% on FPGA performance in NDR and SWI, respectively. Obtained results clearly indicate that FPGA devices perform significantly better with FxBP designs compared to FlBP designs, even with hardened FPUs.**

***Index Terms*—Synthetic Aperture Radar, SAR, Back Projection, FPGA, Hardware Acceleration, OpenCL, HLS, Intel Stratix 10.**

I. Introduction

SYNTHETIC Aperture Radar (SAR) systems are used to obtain high-resolution imagery of various targets. SAR systems have been used by researchers for decades due to their reliable performance in all-weather and all-light conditions. The resolution of the target images depends on the nature (frequency band) of the transmitted signal. Thus, SAR systems can maintain a considerably higher resolution up to thousands of meters or hundreds of kilometers from the target. Unlike optical sensor systems, SAR systems do not depend on environmental conditions. Due to its robustness for environmental conditions, SAR systems are adopted by many applications. Such applications are, monitoring climate change and wildfire progression, military applications, deforestation detection, population estimation, urban planning, and natural disaster monitoring [1]–[5]. To detect targets, first, the SAR system transmits an electromagnetic wave towards the target area. When the system receives a return signal (an echoed signal) from the target, image formation steps are carried out to generate the output image. For the SAR image formation process, two algorithms are used widely. These algorithms can be categorized into two main groups, frequency-domain algorithms and time-domain algorithms. Both algorithm types have their advantages and can be used with specific applications.

Frequency-domain algorithms require less processing power compared to time-domain algorithms. The computational complexity of a frequency-domain algorithm is $O(N^2 log_2 N)$, where N corresponds to the N pulses of echo data and $N \times N$ samples (per image) [6]. Therefore, frequency-domain algorithms such as the Range Doppler (RD) [7]–[9], the Omega-K ($\omega - k$) [10]–[14], and the Chirp Scaling (CS) [15]–[17] algorithms are often used for image formation in SAR systems. However, in general, frequency-domain algorithms are not applicable for all imaging cases due to multiple assumptions that need to be considered for each specific case. Some frequency-domain algorithms rely upon geometric approximations that break on various limits (narrow-angle swath, low squint angle, large image size, wide bandwidth, etc.) while some algorithms assume that the aircraft flies on a linear and uniform trajectory. Frequency-domain algorithms require a motion compensation to reduce the off-track errors generated from non-ideal flight paths. Even though the motion compensation corrects most of the sampling assumptions, it is not perfect, and mostly the compensated value is an approximated value that can yield poorly focused imagery. Also, most frequency-domain algorithms are required to carry out an interpolation in the frequency domain (Stolt interpolation) that can generate artifacts in the output image due to interpolation errors caused by re-sampling data into uniform spacing [18], [19].

Back Projection (BP) is a time-domain algorithm. For the SAR image formation, the BP algorithm uses a matched filter to filter out expected echoed data from an array of target locations. Since the BP method uses the motion compensation implicitly to handle arbitrary flight paths, it does not need a separate module for motion compensation [20]–[22]. Another benefit of the BP is that it can work with all imaging modes (spotlight, stripmap, and scan). Also, the BP algorithm is not affected by the limitations mentioned for the frequency-domain algorithms and shows superiority over the frequency-domain algorithms. However, the BP method is computationally expensive compared to frequency-domain algorithms and requires substantial processing time. The computational complexity of the BP algorithm is $O(N^3)$, where N corresponds to the N pulses of echo data and $N \times N$ samples (per image) [18].

In the past two decades, many BP algorithms have been designed to accelerate the SAR image formation process. Most

The authors are with the Department of Electrical and Computer Engineering, University of Dayton, Dayton, OH, 45469 USA (e-mail: hettiarachchid1@udayton.edu, ebalster1@udayton.edu).

of these methods are derived by modifying the traditional BP algorithm.

Yegulalp et al., introduces a fast back projection (FBP) algorithm [18]. The FBP algorithm introduces a method to divide the synthetic aperture into sub-apertures. Each sub-aperture produces a sub-image later added coherently to generate the final image output. The FBP algorithm use $O(N^{3/2}N_{pulses})$ operations for a $N \times N$ image with N_{pulses} of range compressed data. Compared to the traditional back projection, the FBP algorithm is faster by a factor of $\sqrt{N}$.

Ulander et al., introduces the fast factorized back projection (FFBP) algorithm [23]. The FFBP algorithm is derived from the traditional BP algorithm and by generalizing the FBP algorithm. It uses the Sub-Aperture (SA) concept and each SA uses a Local Polar Coordinate (LPC) system and a pyramid computational architecture to generate full aperture. Then, a fusion technique and 2-D interpolation is used to obtain the final back projected image. Compared to the traditional BP, the FFBP algorithm reduces the computational burden by two to three orders of magnitude.

Zhang et al., introduces an accelerated back projection (AFBP) algorithm [24]. The AFBP algorithm is created by modifying the FFBP algorithm and uses the sub-aperture concept. Instead of the LPC system described in FFBP, the AFBP method use a Unified Polar Coordinate (UPC) system. Also, the AFBP method avoids the 2-D interpolation by converting all data into a 2-D wavenumber domain. In AFBP, sub-aperture fusion is carried out in the 2-D wavenumber domain and finally, all data are converted to the time domain to obtain the back projected image.

In [25], a new method for BP algorithm acceleration using fixed-point arithmetic is described. The method is developed by modifying the BP algorithm described in [26]. Also, the BP algorithm is designed with floating-point arithmetic and fixed-point arithmetic. Both designs are developed with OpenCL and tested on a CPU. It has shown that the fixed-point based BP algorithm gains $\sim 11\%$, $\sim 12\%$, and $\sim 25\%$ for 128×128, 256×256, and 512×512 sized images, respectively. Also, Peak Signal-to-Noise Ratio (PSNR) is calculated for all back projected images to show that the fixed-point based BP algorithm managed to preserve high quality images with 0.2 dB difference.

Even though most of the accelerated BP algorithms are modified from the traditional BP algorithm, few methods use hardware acceleration from devices like General Purpose Graphics Processing Unit (GPGPU) [27]–[30] and Field Programmable Gate Arrays (FPGA) [31], [32]. Since the BP algorithm is highly parallelizable, accelerators can execute multiple operations simultaneously to reduce the processing time significantly. Typically, GPGPUs are used for SAR processing due to their multi-core system. However, GPGPUs require a higher amount of power to maintain their full processing capability. Therefore, GPGPUs may not be a suitable solution with strict space and power requirements.

FPGA devices are capable of processing multiple operations simultaneously with a significantly lower power requirement than GPGPUs. Reconfigurability of FPGA devices can be used for fine-tuning application specific implementations. Compared to functionality, power consumption, and cost, FPGA devices are ranked between GPGPUs and high-end customized Application Specific Integrated Circuits (ASICs). Therefore, FPGA devices may be an ideal choice for SAR processing. Despite the advantages, few BP algorithms are developed using FPGA devices. This is mainly due to two reasons. First, traditional FPGA programming flow and usage of Hardware Description Language (HDL) requires a lengthy programming cycle with simulation and verification. To avoid the tedious HDL programming cycle, many modern FPGA developments use High Level Synthesis (HLS) tools to interpret the hardware designs that are programmed as C/C++ source codes [33], [34]. Secondly, fixed-point design techniques and floating-point design techniques lead to design complications on FPGA devices. Traditionally, FPGA accelerators use fixed-point based designs. Fixed-point based designs are faster and utilize fewer logic resources. However, it is difficult to handle fixed-point based designs with recursive operations on varying dynamic ranges. Due to fixed point scaling, it is hard to create designs with higher accuracy for non-trivial functions like trigonometric and square-root functions [35]. Compared to fixed-point based designs, floating-point based designs are prohibitively costly due to higher logic utilization and slower kernel speed [36]. Designs with higher logic utilization (higher circuit area) require high-end FPGAs, which increase the cost significantly. In 2014, Intel Corporation, introduced single precision hardened Floating-Point Units (FPUs) on DSP blocks with Arria 10 FPGA family [37]. Since FPUs reduce the logic requirement, floating-point based designs are used to process many computationally intensive algorithms on FPGA accelerators.

In [38], it is shown that even with hardened FPUs, floating-point designs that are converted to fixed-point designs use fewer logic resources compared to the floating-point designs. The method is tested with a series of basic arithmetic operations for both floating-point and fixed-point based designs. Also, an FPGA Performance Metric (FPM) is introduced to evaluate the FPGA performance for all designs. The FPM is designed to address the trade-off between logic utilization and kernel frequency (f_{MAX}) caused by the floating-point to fixed-point conversion process. The FPM shows that on average the fixed-point designs have a better FPGA performance than floating-point based designs.

This paper introduces a new Fixed-point based Back Projection (FxBP) algorithm on an FPGA. The FxBP method is developed for spotlight SAR geometry by modifying the Floating-point based Back Projection (FlBP) algorithm described in [25]. The proposed FxBP algorithm is optimized using fixed-point conversions with multiple scale factors and look-up tables. The Open Computing Language (OpenCL) and Intel OpenCL SDK are used to design kernel files for the FPGA. Both FlBP and FxBP kernels are structured as NDRange kernels (FlBP-NDR, FxBP-NDR) and as Single Work Item kernels (FlBP-SWI, FxBP-SWI) to test the FPGA resource utilization and performance for different design techniques. All designs are tested with BittWare 520N-MX board which is equipped with an Intel Stratix 10-MX2100 FPGA. Intel Quartus Pro/AOCL 19.3 version is used to compile and generate FPGA bit-stream. The Board Support Package (BSP) developed by BittWare (BIST 1.0-1) is used to program the FPGA.

Comparing the results obtained from the FxBP and FlBP algorithm designs on Stratix 10 FPGA device shows that FPGA performance on FxBP designs is higher than FlBP designs. The FxBP-NDR show a gain of 17.90% in performance increase than FlBP-NDR and the FxBP-SWI show a gain of 91.62% in performance increase than FlBP-SWI. Also, FxBP designs use fewer resources compared to FlBP designs. Compared to FlBP designs, the FxBP-NDR and FxBP-SWI save 37.47% and 44.06% resources on ALUTs, 16.76% and 23.83% resources on Registers, and 16.87% and 42.54% resources on Logic Utilization, respectively. All designs are tested with synthetic video phase history data (VPH) generated from [26] and real phase history data from the GOTCHA dataset [39]. Back projected image quality is calculated using Peak signal-to-noise ratio (PSNR), and it shows that FxBP designs manage to preserve image quality while using fewer resources compared to FlBP designs.

Following the introduction, Section II consists an overview Spotlight SAR Model and BP implementation. Section III introduces the fixed-point based BP algorithm and its implementation. Section IV describes the FPGA programming model used to design and program BP algorithms. Section V consists obtained results, and analysis and Section VI is dedicated for conclusion.

II. Spotlight SAR Model

This section describes a brief overview of the SAR image processing modules introduced in the spotlight SAR simulator [26]. The spotlight SAR simulator is used to generate a synthetic phase history (echoed data) for digital images. In order to obtain the phase history data, a simulated aircraft rotating around the target (digital image) is used. Figure1 shows the SAR image formation flow. Image formation modules receive the phase history (echoed data) of the target and generate a range profile. Then the range profile is used by the back projection module to generate the final back projected image output.

A spotlight SAR imaging model [25] shown in Figure 2 is used for designing floating-point based BP and fixed-point based BP algorithms. The range profile generation from phase history is described briefly for completeness.

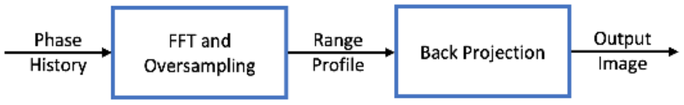


Figure 1: SAR image formation flow [26]

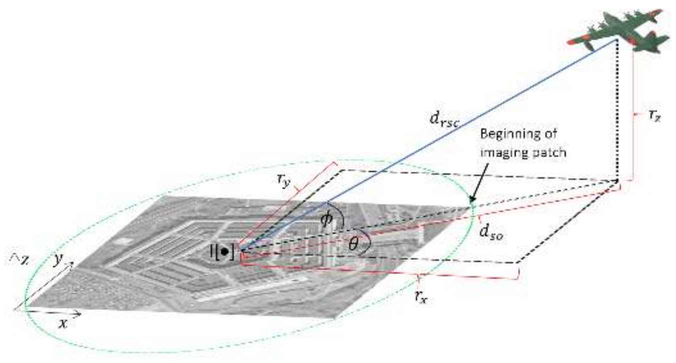


Figure 2: Spotlight SAR imaging model for floating-point processing [25]

A. Generate Range Profile

First, the distance between radar to each pixel $d_{ac}[x, y, \theta]$ of the image is shown in (1).

$$d_{ac}[x, y, \theta] = \sqrt{(xG - d_{so}\sin(\theta))^2 + (yG - d_{so}\cos(\theta))^2 + r_x^2}, \tag{1}$$

where $x(\in [\frac{N}{2}, \frac{-N}{2}])$ and $y(\in [\frac{N}{2}, \frac{-N}{2}])$ are row and column image indexes, N is the number of rows and columns, G is the ground sample distance (GSD), and θ is the horizontal angular displacement. Then the distance between scene center and each pixel is given by,

$$d[x, y, \theta] = d_{ac}[x, y, \theta] - \sqrt{d_{alt}^2 + \left(d_{so} - \frac{NG}{\sqrt{2}}\right)^2}. \tag{2}$$

The distance vector $d[\cdot]$ in (2) is used to calculate the echoed pulse return $x_{ret}(t, \theta)$. As shown in (3), $x_{ret}(t, \theta)$ is derived by summing up all pulse returns from each individual pixel distances.

$$x_{ret}(t, \theta) = \sum_{x,y} I[x, y] x_p \left(t - \frac{2d[x, y, \theta]}{c}\right) \tag{3}$$

where the $x_p(t)$ is the transmitted linear frequency modulated pulse, and c is the speed of light. To apply matched filtering operation, first, the $x_{ret}(t, \theta)$ is demodulated then sampled and discretized. As shown in (4), demodulation is done by mixing the $x_{ret}(t, \theta)$ with carrier signal $(c(t))$ and applying a low-pass filter.

$$x_{mix}(t, \theta) = F\{x_{ret}(t, \theta)c(t)\}, \tag{4}$$

where $F\{\cdot\}$ is the LPF operator, $c(t) = cos(2\pi f_l t)$ and f_l is the lowest frequency of the chirp signal. Then the demodulated $x_{mix}(t, \theta)$ is sampled and discretized to create

$$x_{mix}[n, \theta] = x_{mix}(t, \theta)|_{t=nT}, \tag{5}$$

where, $T = (2S_0 BW_p)^{-1}$, the S_0 is an oversampling factor, $BW_p = 2f_m T_p$ is the bandwidth, f_m is the modulation frequency, and T_p represents the duration of the pulse signal. The Match filter process in the frequency domain is given by,

$$X_{ph}[n,\theta] = X_{mix}[n,\theta] H_{mf}[n], \tag{6}$$

where $H_{mf}[n]$ is the frequency response of the matched filter. Finally, the range profile ($R_p[k,\theta]$) is obtained by applying FFT and oversampling to the phase history. The range profile is given by,

$$R_p[k,\theta] = \sum_n X_{ph}[n,\theta] e^{-j\frac{2\pi n}{N}k}, \tag{7}$$

where N represents the number of samples in the FFT.

B. Back Projection

The back projection module use the $R_p[k,\theta]$ as an input and back projected data into a 2-D image space to generate the final output image. First, differential range $d_R[\xi]$ is calculated by,

$$d_R[\xi] = d_{rcp}[\xi] - d_{rsc}[\xi];\ \xi \in (x, y, z) \tag{8}$$

where, $d_{rcp}[\xi]$ is the distance between radar to current pixel, and $d_{rsc}[\xi]$ is the distance from radar to scene center. The $d_{rcp}[\xi]$) is given by,

$$d_{rcp}[\xi] = \sqrt{(r_x - d_x)^2 + (r_y - d_y)^2 + r_z^2}, \tag{9}$$

where r_x, r_y, r_z are ranges from radar to imaging scene, and d_x, d_y are the displacements from scene center to current pixel. The $d_{rsc}[\xi]$ is shown in (10).

$$d_{rsc}[\xi] = \sqrt{r_x^2 + r_y^2 + r_z^2} \tag{10}$$

After calculating the $d_R[\xi]$, back projection module applies a phase correction and projects the range profile data to the 2-D image space. Then for each return signal, projected grids are summed up to generate the final image output. Back projected image output is obtained by,

$$\tilde{I}[x,y] = \frac{1}{\Xi_N}\left|\sum_\xi \tilde{R}_p(i,\xi) e^{-j\gamma[\xi]}\right|,\ \ \gamma[\xi] = \frac{4\pi f_l d_R[\xi]}{c}, \tag{11}$$

where Ξ_N is the total number of projected range profiles, $e^{-j\gamma[\xi]}$ is the phase correction term and $\tilde{R}_p(i,\xi)$ is a bi-linearly interpolated range profile shown in (12). The range profile is interpolated to convert discrete range profile data to continuous data.

$$\begin{aligned}\tilde{R}_p(i,\xi) =& \{(\lfloor i+1\rfloor - i) R_p[\lfloor i\rfloor, \xi] + \\ & (i - \lfloor i\rfloor) R_p[\lfloor i+1\rfloor, \xi]\},\end{aligned} \tag{12}$$

and i is given by $i = d_R[\xi]/d_s$ and d_s is the sample distance.

C. OpenCL Implementation of the FlBP Algorithm

The Floating-point based Back Projection (FlBP) OpenCL implementations are shown in Listing 1 and 2. The FlBP kernel uses OpenCL N-Dimensional Range (NDR) structure (FlBP-NDR) as shown in Listings 1. In FlBP-NDR method, each pixel of the BP image gets assigned as an independent work item. To optimize the process, compiler executes multiple work items concurrently (data parallelism). The FlBP kernel uses OpenCL Single Work Item (SWI) structure (FlBP-SWI) as shown in Listings 2. In the FlBP-SWI method, single work item is assigned to generate the BP image output for a single return pulse. The compiler optimizes the process by using pipeline techniques (task parallelism).

In Listing 1, lines 1-4 initialize the kernel and variables. The back projected image output in Equation (11) shows that the BP modules require complex math calculations. Therefore, range profile ($R_p[\cdot]$) and image output ($\tilde{I}[\cdot]$) are divided to real ($R_{pr}, \tilde{I}_r$) and imaginary ($R_{pi}, \tilde{I}_i$) samples for calculation. In Line 5, current pixel index of the BP image (u) is obtained. Line 6 and 7, initialize temporary image variables for real (Imr) and imaginary (Imi) samples. Lines 8 and 9 calculates the (x, y) positions of the image and lines 10 and 11 show the displacement from current pixel to scene center in (x, y) directions, where C ($\in 128, 256, 512$) represents the number of columns and rows in the image and G is the ground sampling distance. Line 12 calculates the intensity values for all pixels, where N represents the number of pulses (image size). Lines 13-15 show the differential range ($d_R[\xi]$) calculation. Lines 16-17 shows the calculation of nearest range profile sample indexes, where P represents the number of FFT samples. Line 18 constrains the sample index k within the range profile. Line 19 shows the calculation of phase correction term (γ), where $B = 4\pi/c$. The d in line 20 represents the differential distance from current pixel to scene center and the t in line 21 represents the difference between current pixel to scene center and the nearest range profile sample. Lines 22-25 show the bi-linear interpolation of range profile. Lines 26-27 show the `sine` and `cosine` angle calculation for phase correction term (γ). Lines 28-29 show the summed up real and imaginary intensity values for all range profiles. Finally, Lines 31-32, show real and imaginary samples of the back projected image output.

The FlBP-SWI in Listing 2 is very similar to the FlBP-NDR in Listing 1. The only difference is in the line 5, where in FlBP-NDR method, the variable u represents the image pixel index and in FlBP-SWI method, variable u is the index that used to calculate intensities for all pixels in the image.

III. Fixed-Point Based Back Projection Algorithm

The Fixed-point based Back Projection (FxBP) algorithm is designed by converting back projection variables listed in Section II-B into integers using fixed-point arithmetic. Due to rounding-off, typical floating-point to fixed-point conversion generates data with less accuracy. However, multiplying the floating-point variable with a constant scale value and then

Listing 1 : The FlBP-NDR OpenCL module

```
kernel void FlBP_NDR $(\tilde{I}[\cdot], R_p[\cdot,\xi], P, N, d_s, r_x, r_y, r_z, G, f_l, C)\{$
    float $d_x, d_y, d_{rsc}, d_{rcp}, d_R, \gamma, d, t, S_\gamma, C_\gamma$;
    float $I_{mr}, I_{mi}, \tilde{R}_{pr}, \tilde{R}_{pi}$;
    int $x, y, k, m, ml$;
    $u =$ get_global_id$(0)$;
    $I_{mr} = 0$;
    $I_{mi} = 0$;
    $x = u\%C$;
    $y =$ (int)$(u/C)$;
    $d_x = ($(float)$y-$(float)$C/2 + 0.5)G$;
    $d_y = ($(float)$C/2-$(float)$x/2 - 0.5)G$;
    for$(n = 0; n < N; n++)\{$
        $d_{rsc} =$ sqrt$(r_x^2[n] + r_y^2[n] + r_z^2[n])$;
        $d_{rcp} =$ sqrt$((r_x[n]-d_x)^2+(r_y[n]-dy)^2+r_z^2[n])$;
        $d_R = d_{rsc} - d_{rcp}$;
        $k =$ (int)$(d_R/d_S) + ((P+1) >> 1)$;
        if$(d_R >= 0)$ $k+ = 1$;
        if$(k > 0$ && $k < P)\{$
            $\gamma = Bf_l d_R$;
            $d = d_S(k - ((P+1) >> 1))$;
            $t = d - d_R$;
            $m = kN + n$;
            $ml = m - N$;
            $\tilde{R}_{pr} = (R_{pr}[ml]t + R_{pr}[m](d_S - t))/d_S$;
            $\tilde{R}_{pi} = (R_{pi}[ml]t + R_{pi}[m](d_S - t))/d_S$;
            $S_\gamma =$ sin$(\gamma)$;
            $C_\gamma =$ cos$(\gamma)$;
            $I_{mr} += \tilde{R}_{pr}C_\gamma - \tilde{R}_{pi}S_\gamma$;
            $I_{mi} += \tilde{R}_{pr}S_\gamma + \tilde{R}_{pi}C_\gamma$;
        $\}$
    $\}$
    $\tilde{I}_r[u] = I_{mr}/N$;
    $\tilde{I}_i[u] = I_{mi}/N$;
$\}$
```

Listing 2 : The FlBP-SWI OpenCL module

```
kernel void FlBP_SWI $(\tilde{I}[\cdot], R_p[\cdot,\xi], P, N, d_s, r_x, r_y, r_z, G, f_l, C)\{$
    float $d_x, d_y, d_{rsc}, d_{rcp}, d_R, \gamma, d, t, S_\gamma, C_\gamma$;
    float $I_{mr}, I_{mi}, \tilde{R}_{pr}, \tilde{R}_{pi}$;
    int $x, y, k, m, ml$;
    for$(u = 0; u < C^2; u++)\{$
        $I_{mr} = 0$;
        $I_{mi} = 0$;
        $x = u\%C$;
        $y =$ (int)$(u/C)$;
        $d_x = ($(float)$y-$(float)$C/2 + 0.5)G$;
        $d_y = ($(float)$C/2-$(float)$x/2 - 0.5)G$;
        for$(n = 0; n < N; n++)\{$
            $d_{rsc} =$ sqrt$(r_x^2[n] + r_y^2[n] + r_z^2[n])$;
            $d_{rcp} =$ sqrt$((r_x[n] - d_x)^2 + (r_y[n] - dy)^2 + r_z^2[n])$;
        $d_R = d_{rsc} - d_{rcp}$;
        $k =$ (int)$(d_R/d_S) + ((P+1) >> 1)$;
            if$(d_R >= 0)$ $k+ = 1$;
            if$(k > 0$ && $k < P)\{$
                $\gamma = Bf_l d_R$;
                $d = d_S(k - ((P+1) >> 1))$;
                $t = d - d_R$;
                $m = kN + n$;
                $ml = m - N$;
                $\tilde{R}_{pr} = (R_{pr}[ml]t + R_{pr}[m](d_S - t))/d_S$;
                $\tilde{R}_{pi} = (R_{pi}[ml]t + R_{pi}[m](d_S - t))/d_S$;
                $S_\gamma =$ sin$(\gamma)$;
                $C_\gamma =$ cos$(\gamma)$;
                $I_{mr} += \tilde{R}_{pr}C_\gamma - \tilde{R}_{pi}S_\gamma$;
                $I_{mi} += \tilde{R}_{pr}S_\gamma + \tilde{R}_{pi}C_\gamma$;
            $\}$
        $\}$
    $\tilde{I}_r[u] = I_{mr}/N$;
    $\tilde{I}_i[u] = I_{mi}/N$;
    $\}$
$\}$
```

converting it to fixed-point variable can preserve high accuracy of the floating-point variable. Higher scale values can generate high accuracy from the conversion process. After processing fixed-point operations, fixed-point data can be converted back to floating-point data by dividing with the same scale factor. Scale values are limited to power of 2 to increase the design optimization by applying binary shifts for multiplications and division operations [40]. Floating-point to fixed-point conversion is represented by,

$$\hat{F} = \lfloor 2^{\lambda} F \rfloor, \tag{13}$$

where F represents the floating-point variable, $\hat{F}$ is the fixed-point variable, and λ is the scale factor. The Table I shows the floating-point variables, scale factors, and fixed-point variables generated for BP algorithm described in Section II-B. All floating-point variables are converted using Equation 13, and converted fixed-point variables are represented with a `hat` symbol.

Table I: Floating-point variable to fixed-point variable conversion map with scale factors.

Floating-point variables	Scale factor	Fixed-point variables
r_x, r_y, r_z	2^{λ_R}	$\hat{r}_x, \hat{r}_y, \hat{r}_z$
d_x, d_y, d_z		$\hat{d}_x, \hat{d}_y, \hat{d}_z$
d_R		$\hat{d}_R$
d_{rcp}		$\hat{d}_{rcp}$
d_{rsc}		$\hat{d}_{rsc}$
$\tilde{I}$		$\hat{\tilde{I}}$
γ		$\hat{\gamma}$
R_p	2^{λ_M}	$\hat{R}_p$
S_γ	2^{β}	$\hat{S}_\gamma$
C_γ		$\hat{C}_\gamma$

There are three scale factors defined in Table I, λ_R, λ_M, and β. The λ_R scale is applied to variables that pose a significant impact for the overall accuracy, such as range distances (r_x, r_y, r_z) and distances from radar to scene center (d_{rsc}) or current pixel (d_{rsc}). The λ_M scale is used for range profile (R_p) conversion. Typically, range profile consists significantly smaller values compared to range distances. Method described in [25] is applied to find the optimum λ_R, λ_M values that can generate higher quality output. Both λ_R and λ_M are varied from 1 to 31 and parameters such as standoff distance(d_{so}), altitude (r_z),and patch width (target area) are varied to increase the robustness of the algorithm. Obtained results for all configurations are averaged to find an optimum range for λ_R and λ_M values. The BP algorithm generates highest PSNR values for image output when λ_R is at 15 and when λ_M varies between 14-16. Therefore, both λ_R and λ_M are set as 15 for the BP algorithm. The β scale value is used for the conversion of phase correction $(\gamma[\cdot])$, that consists angle values obtained from `Sine` and `Cosine` functions. The angle conversion is explained in Section III-A2.

A. *Fixed-Point Conversion*

The fixed-point conversion of the BP algorithm is divided into two main parts. First, the fixed-point based differential range $(\hat{d}_R[\xi])$ is calculated. Secondly, the fixed-point based conversion is applied to the phase correction term $(\hat{\gamma}[\xi])$. Finally, the BP algorithm equations are redefined with fixed-point variables.

1) Differential Range Conversion: The differential range $(d_R[\xi])$ in Equation (8) is converted to a fixed-point vector. The converted differential range $(\hat{d}_R[\xi])$ is given by,

$$\hat{d}_R[\xi] = \hat{d}_{rcp}[\xi] - \hat{d}_{rsc}[\xi], \qquad \xi \in (x, y, z), \tag{14}$$

where,

$$\hat{d}_{rcp}[\xi] = \left\lfloor \sqrt{(\hat{r}_x - \hat{d}_x)^2 + (\hat{r}_y - \hat{d}_y)^2 + \hat{r}_z^2} \right\rfloor, \tag{15}$$

$$\hat{d}_{rsc}[\xi] = \left\lfloor \sqrt{\hat{r}_x^2 + \hat{r}_y^2 + \hat{r}_z^2} \right\rfloor. \tag{16}$$

The $\hat{r}_x, \hat{r}_y, \hat{r}_z$ are converted fixed-point ranges from radar to imaging scene, and $\hat{d}_x, \hat{d}_y$ are the fixed-point displacements from scene center to current pixel.

2) Phase Correction Conversion: Converting an exponential term with varying dynamic range, to a fixed-point variable is a challenging task that requires higher amount of logic resources. Therefore, Euler's formula is used to convert the phase correction term $(e^{-j\gamma[\xi]})$ in Equation (11) to a fixed-point variable. Using Euler's formula, $e^{-j\hat{\gamma}[\xi]}$ is redefined as shown in Equation (17).

$$e^{-j\hat{\gamma}} = \hat{C}_\gamma - j\hat{S}_\gamma, \tag{17}$$

where $\hat{S}_\gamma = sin(\hat{\gamma}[\xi])$, $\hat{C}_\gamma = cos(\hat{\gamma}[\xi])$, and $\hat{\gamma}$ is given by,

$$\hat{\gamma} = \frac{4\pi f_l \hat{d}_R[\xi]}{c}. \tag{18}$$

In the FlBP method, `sine` and `cosine` angles are calculated using OpenCL library functions. However, for the FxBP method, library functions can generate inaccurate results with scaled up fixed-point angle values. Therefore, a fixed-point look-up table is created to calculate the $\hat{S}_\gamma$ and $\hat{C}_\gamma$ angle values. First, 2π range is divided to a 2^β number of linearly spaced data array. The array indexes are given by,

$$\theta[i] = \frac{2\pi}{2^\beta} i, \tag{19}$$

where $\theta[i]$ is the i^{th} data point of the array, and 2^β is the scale. As shown in (20), the scaled-up fixed-point look-up table is created by calculating `sine` angles for each data point and multiplying with the scale factor of 2^β.

$$F_{sin}[i] = \lfloor \mathtt{sin}(\theta[i]) \times 2^\beta \rfloor, \tag{20}$$

where $F_{sin}[\cdot]$ is the look-up table with 2^β samples for one full period of the `sin()` function. Same method can be used to create a look-up table for `cosine` angles $(F_{cos}[\cdot])$. Approximated `sine` and `cosine` values can be calculated by,

$$\widetilde{sin}[\theta[i]] = \frac{F_{sin}[i]}{2^\beta}, \widetilde{cos}[\theta[i]] = \frac{F_{cos}[i]}{2^\beta}, \tag{21}$$

where $i = \lfloor 2^\beta \theta / 2\pi \rfloor$. Figure 3 shows a comparison between approximated `sine` and `cosine` values and `sin()` and `cos()` library functions. Accuracy of the approximated `sine` and `cosine` values can be increased by increasing the 2^β scale.

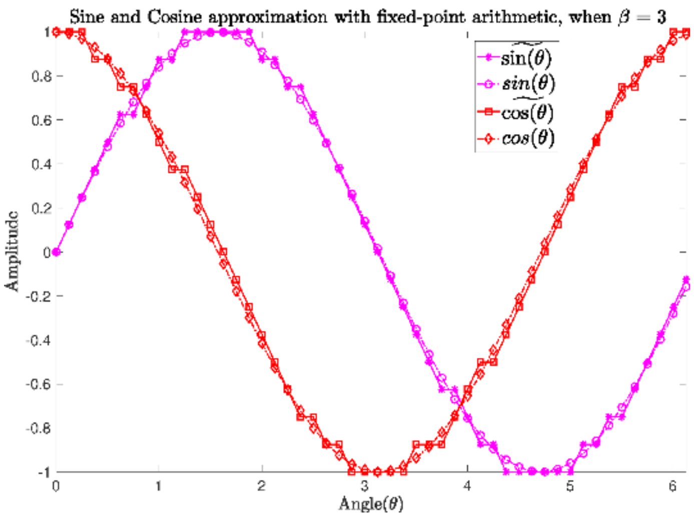


Figure 3: Sine and Cosine approximation with fixed-point arithmetic

In the FxBP algorithm, resources are saved by only using $F_{sin}[\cdot]$ look-up table in the design and `sine` angle $\hat{S}_\gamma$ is calculated by,

$$\hat{S}_\gamma = \hat{F}_{sin}[\hat{\gamma}] 2^{(\lambda_R - \beta)}. \tag{22}$$

The `cosine` angle $\hat{C}_\gamma$ is calculated by shifting $F_{sin}[\cdot]$ look-up table by a quarter of a period and is given by,

$$\hat{C}_\gamma = \hat{F}_{sin}[(\hat{\gamma} + \frac{2^\beta}{4})\%2^\beta] 2^{(\lambda_R - \beta)}, \tag{23}$$

where $\%$ is the modulus operator. The modulus operator is used to find the principle angle in the 2π scale. However, modulus and division operators use higher number of logic resources. Thus, the Equation (23) is redefined with bit-wise `and` operator and with bit-shift operator and it is given by,

$$\hat{C}_\gamma = \hat{F}_{sin}[(\hat{\gamma} + (2^\beta >> 2))\&(2^\beta - 1)] << (\lambda_R - \beta). \quad (24)$$

The $\hat{S}_\gamma$ and $\hat{C}_\gamma$ are multiplied with $2^{(\lambda_R-\beta)}$ scale to match the range profile scale.

B. The BP Algorithm With Fixed-Point Variables

As described in Section III-A1 and III-A2, once $\hat{d}_R[\xi]$ and $e^{-j\hat{\gamma}[\xi]}$ are calculated, redefined Equation (11) for back projected image output is given by,

$$\hat{\hat{I}}[x,y] = \frac{1}{\Xi_N}\left|\sum_\xi \hat{\hat{R}}_p(i,\xi)e^{-j\hat{\gamma}[\xi]}\right|, \ \ \hat{\gamma}[\xi] = \frac{4\pi f_l \hat{d}_R[\xi]}{c}, \quad (25)$$

where,

$$\begin{aligned}\hat{\hat{R}}_p(i,\xi) = &\{(\lfloor i+1\rfloor - i)\hat{R}_p[\lfloor i\rfloor,\xi] + \\ &(i - \lfloor i\rfloor)\hat{R}_p[\lfloor i+1\rfloor,\xi]\}2^{(\lambda_R-\lambda_M)},\end{aligned} \quad (26)$$

and i is given by

$$i = \frac{\hat{d}_R[\xi]}{\hat{d}_s}. \quad (27)$$

The $\hat{R}_p(i,\xi)$ in (26) is scaled up with 2^{λ_M} and then the $\hat{\hat{R}}_p(i,\xi)$ is scaled up by $2^{\lambda_R-\lambda_M}$ to match the $e^{-j\hat{\gamma}[\xi]}$ scale. Finally, as shown in Equation (28), the scaled-up back projected image output ($\hat{\hat{I}}[x,y]$) is scaled-down to obtain the final image output.

$$\tilde{I}[x,y] = \frac{\hat{\hat{I}}[x,y]}{2^{\lambda_R}} \quad (28)$$

Listing 3 : The FxBP-NDR OpenCL module

```
1: kernel void FxBP_NDR ($\hat{\hat{I}}[\cdot], \hat{R}_p[\cdot,\xi], P, N, \hat{d}_s, \hat{r}_x,$
   $\hat{r}_y, \hat{r}_z, \hat{G}, f_l, C)\{$
2:    int $x, y, k, m, ml, \hat{S}_\gamma, \hat{C}_\gamma, \hat{\hat{R}}_{pr}, \hat{\hat{R}}_{pi}$;
3:    long $\hat{I}_{mr}, \hat{I}_{mi}, \hat{d}_{rsc}, \hat{d}_{rcp}, \hat{d}_R, d, t$;
4:    long $\hat{d}_x, \hat{d}_x, \hat{r}_{xl}, \hat{r}_{yl}, \hat{r}_{zl}, \hat{r}_{xd}, \hat{r}_{yd}, \hat{r}_{zd}$;
5:    constant $i_C$ =(int)$(1 << \lambda_R)/C$;
6:    constant $i_K$ =(int)$(1 <<$kScale$)/d_S$;
7:    constant $i_D$ =(int)$(1 << \lambda_R)/d_S$;
8:    constant $i_N$ =(int)$(1 << \lambda_R)/N$;
9:    $u =$ get_global_id(0);
10:   $\hat{I}_{mr} = 0$;
11:   $\hat{I}_{mi} = 0$;
12:   $x = u\%C$;
13:   $y = ui_C >> \lambda_R$;
14:   $\hat{d}_x =$ (long)$(2y - C + 1)\hat{G}$;
15:   $\hat{d}_y =$ (long)$(C - 2x - 1)\hat{G}$;
16:   for$(n = 0; n < N; n++)\{$
17:      $\hat{r}_{xl}$ =(long)$\hat{r}_x[n]$;
18:      $\hat{r}_{yl}$ =(long)$\hat{r}_y[n]$;
19:      $\hat{r}_{zl}$ =(long)$\hat{r}_z[n]$;
20:      $\hat{r}_{xd} = \hat{r}_{xl} - \hat{d}_x$;
21:      $\hat{r}_{yd} = \hat{r}_{yl} - \hat{d}_y$;
22:      $\hat{r}_{zd} = \hat{r}_{zl}^2$;
23:      $\hat{d}_{rsc} =$ (long)sqrt((float)$(\hat{r}_{xl}^2 + \hat{r}_{yl}^2 + \hat{r}_{zl}^2)$);
24:      $\hat{d}_{rcp} =$ (long)sqrt((float)$(\hat{r}_{xd}^2 + \hat{r}_{yd}^2 + \hat{r}_{zd})$);
25:      $\hat{d}_R = \hat{d}_{rsc} - \hat{d}_{rcp}$;
26:      $k = ((\hat{d}_R i_K) >>$kScale$) + ((P+1) >> 1)$;
27:      if$(\hat{d}_R >= 0)\ k += 1$;
28:      if$(k > 0\ \&\&\ k < P)\{$
29:         $\hat{\gamma} = (\hat{B}\hat{d}_R) >> \lambda_R$;
30:         $d = \hat{d}_S(k - ((P+1) >> 1))$;
31:         $t = d - \hat{d}_R$;
32:         $m = kN + n$;
33:         $ml = m - N$;
34:         $\hat{\hat{R}}_{pr} = (\hat{R}_{pr}[ml]t + \hat{R}_{pr}[m](\hat{d}_S - t))i_D >> \lambda_R$;
35:         $\hat{\hat{R}}_{pi} = (\hat{R}_{pi}[ml]t + \hat{R}_{pi}[m](\hat{d}_S - t))i_D >> \lambda_R$;
36:         $\hat{S}_\gamma = \hat{F}_{sin}[\hat{\gamma}] << (\lambda_R - \beta)$;
37:         $\hat{C}_\gamma = \hat{F}_{sin}[(\gamma + (2^\beta >> 2))\&(2^\beta - 1)] <<$
38:               $(\lambda_R - \beta)$;
39:         $\hat{I}_{mr} += (\hat{\hat{R}}_{pr}\hat{C}_\gamma - \hat{\hat{R}}_{pi}\hat{S}_\gamma) >> \lambda_M$;
40:         $\hat{I}_{mi} += \hat{\hat{R}}_{pr}\hat{S}_\gamma + \hat{\hat{R}}_{pi}\hat{C}_\gamma >> \lambda_M$;
41:      $\}$
42:   $\}$
43:   $\hat{\hat{I}}_r[u] = \hat{I}_{mr} i_N >> \lambda_R$;
44:   $\hat{\hat{I}}_i[u] = \hat{I}_{mi} i_N >> \lambda_R$;
45: $\}$
```

C. OpenCL Implementation of the FxBP Algorithm

The FxBP-NDR and FxBP-NDR OpenCL kernels are shown in Listings 3 and 4. The FxBP-SWI is very similar to the FxBP-NDR. The only difference is in the line 9, where in FxBP-NDR method, the variable u represents the image pixel index and in FxBP-SWI method, variable u is the index that is used to calculate intensities for all pixels in the image.

In Listing 3, lines 1-4 initialize the kernel and fixed-point variables. Lines 5-8 show constant values that are used to avoid repeated division operation in the kernel. Instead of dividing a variable with a denominator, the variable is multiplying with scaled up inverse denominator. In line 9, current pixel index of the BP image (u) is obtained. Lines 10 and 11, initialize temporary fixed-point image variables for real ($\hat{I}mr$) and imaginary ($\hat{I}mi$) samples. Lines 12 and 13 calculates the (x,y) positions of the image. The modulus operation in FlBP-NDR method is replaced with bitwise modulus operation to calculate x position and y is calculated using constant inverse denominator value to avoid the costly division operation. Lines 14-15 show the displacement from current pixel to scene center in (x,y) directions, where C $(\in 128, 256, 512)$ represents the number of columns and rows in the image and $\hat{G}$ is the fixed-point ground sampling distance. Line 16 calculates the intensity values for all pixels, where N represents the number of pulses (image size). Lines 17-25 show the differential range ($\hat{d}_R[\xi]$) calculation. Lines 26-27 shows the calculation of nearest range profile sample indexes (k), where `kScale` is a constant scale value and P represents the number of FFT samples. Line 28 constrains the k within the range profile. Line 29 shows the calculation of phase correction term ($\hat{\gamma}$), where $\hat{B} = 4\pi f_l/c$. The $\hat{\gamma}$ scaled down by λ_R to avoid double scale up. The d in line 30 represents the differential distance from current pixel to scene center and the t in line 31 represents the difference between current pixel to scene center and the nearest range profile sample. Lines 32-35 show the bi-linear interpolation of the fixed-point based range profile. The range profile real ($\hat{\hat{R}}_{pr}$) and imaginary ($\hat{\hat{R}}_{pi}$) samples are scaled down by λ_R to avoid double scale up. Lines 36-37 show the `sine` and `cosine` angle calculation for fixed-point based phase correction term ($\hat{\gamma}$). As described in Equation (22), the $\hat{S}_\gamma$ and $\hat{C}_\gamma$ are scaled up by ($\lambda_R - \beta$) to match the range profile scale. Lines 38-39 show the summed up real ($\hat{I}_{mr}$) and imaginary ($\hat{I}_{mi}$) intensity values for all range profiles and both of them are scaled down by λ_M to remove λ_M scale from all the image variables. Finally, Lines 41-42, show the scaled up real ($\hat{I}_r$) and imaginary ($\hat{I}_i$) samples of the back projected image output.

IV. FPGA Programming Model

The FPGA programming model consists two main coding paths, host side coding path and device side coding path. The Figure 4 shows the FPGA Programming Model used to design FlBP and FxBP algorithms.

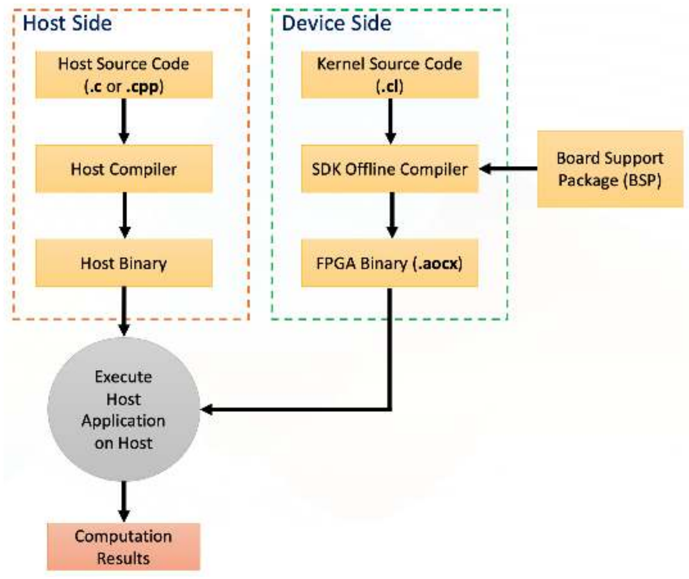


Figure 4: FPGA Programming Model

As shown in Figure 4, the host side is typically designed with the C++ language and OpenCL C++ wrapper API. The source code is compiled using the system compiler to generate the host binary code. The device side source code is designed using OpenCL 1.2 and Intel OpenCL SDK is used to compile the FPGA binary (.aocx file). The manufacturer supplied board support package (BSP) is integrated to the SDK to communicate and program the device. Once the offline compiler generates the FPGA binary, the host can

Listing 4 : The FxBP-SWI OpenCL module

```
1:  kernel void FxBP_SWI ($\hat{\hat{I}}[\cdot], \hat{R}_p[\cdot,\xi], P, N, \hat{d}_s, \hat{r}_x,$
    $\hat{r}_y, \hat{r}_z, \hat{G}, f_l, C$){
2:      int $x, y, k, m, ml, \hat{S}_\gamma, \hat{C}_\gamma, \hat{\hat{R}}_{pr}, \hat{\hat{R}}_{pi}$;
3:      long $\hat{I}_{mr}, \hat{I}_{mi}, \hat{d}_{rsc}, \hat{d}_{rcp}, \hat{d}_R, d, t$;
4:      long $\hat{d}_x, \hat{d}_x, \hat{r}_{xl}, \hat{r}_{yl}, \hat{r}_{zl}, \hat{r}_{xd}, \hat{r}_{yd}, \hat{r}_{zd}$;
5:      constant $i_C =$(int)$(1 << \lambda_R)/C$;
6:      constant $i_K =$(int)$(1 <<$kScale$)/d_S$;
7:      constant $i_D =$(int)$(1 << \lambda_R)/d_S$;
8:      constant $i_N =$(int)$(1 << \lambda_R)/N$;
9:      $u =$ get_global_id(0);
10:     for($u = 0; u < C^2; u++$){
11:         $\hat{I}_{mr} = 0$;
12:         $\hat{I}_{mi} = 0$;
13:         $x = u\%C$;
14:         $y = ui_C >> \lambda_R$;
15:         $\hat{d}_x =$ (long)$(2y - C + 1)\hat{G}$;
16:         $\hat{d}_y =$ (long)$(C - 2x - 1)\hat{G}$;
17:         for($n = 0; n < N; n++$){
18:             $\hat{r}_{xl} =$(long)$\hat{r}_x[n]$;
19:             $\hat{r}_{yl} =$(long)$\hat{r}_y[n]$;
20:             $\hat{r}_{zl} =$(long)$\hat{r}_z[n]$;
21:             $\hat{r}_{xd} = \hat{r}_{xl} - \hat{d}_x$;
22:             $\hat{r}_{yd} = \hat{r}_{yl} - \hat{d}_y$;
23:             $\hat{r}_{zd} = \hat{r}_{zl}^2$;
24:             $\hat{d}_{rsc} =$ (long)sqrt((float)($\hat{r}_{xl}^2 + \hat{r}_{yl}^2 +$
25:                 $\hat{r}_{zl}^2$));
26:             $\hat{d}_{rcp} =$ (long)sqrt((float)($\hat{r}_{xd}^2 + \hat{r}_{yd}^2 +$
27:                 $\hat{r}_{zd}$));
28:             $\hat{d}_R = \hat{d}_{rsc} - \hat{d}_{rcp}$;
29:             $k = ((\hat{d}_R i_K) >>$kScale$) + ((P + 1) >> 1)$;
30:             if($\hat{d}_R >= 0$) $k += 1$;
31:             if($k > 0$ && $k < P$){
32:                 $\hat{\gamma} = (\hat{B}\hat{d}_R) >> \lambda_R$;
33:                 $d = \hat{d}_S(k - ((P + 1) >> 1))$;
34:                 $t = d - \hat{d}_R$;
35:                 $m = kN + n$;
36:                 $ml = m - N$;
37:                 $\hat{\hat{R}}_{pr} = (\hat{R}_{pr}[ml]t + \hat{R}_{pr}[m](\hat{d}_S - t))i_D >>$
38:                     $\lambda_R$;
39:                 $\hat{\hat{R}}_{pi} = (\hat{R}_{pi}[ml]t + \hat{R}_{pi}[m](\hat{d}_S - t))i_D >> \lambda_R$;
40:                 $\hat{S}_\gamma = \hat{F}_{sin}[\hat{\gamma}] << (\lambda_R - \beta)$;
41:                 $\hat{C}_\gamma = \hat{F}_{sin}[(\gamma + (2^\beta >> 2))\&(2^\beta - 1)] <<$
42:                     $(\lambda_R - \beta)$;
43:                 $\hat{I}_{mr} += (\hat{\hat{R}}_{pr}\hat{C}_\gamma - \hat{\hat{R}}_{pi}\hat{S}_\gamma) >> \lambda_M$;
44:                 $\hat{I}_{mi} += \hat{\hat{R}}_{pr}\hat{S}_\gamma + \hat{\hat{R}}_{pi}\hat{C}_\gamma >> \lambda_M$;
45:             }
46:         }
47:         $\hat{\hat{I}}_r[u] = \hat{I}_{mr} i_N >> \lambda_R$;
48:         $\hat{\hat{I}}_i[u] = \hat{I}_{mi} i_N >> \lambda_R$;
49:     }
50: }
```

execute the host binary to send and receive data from the device.

Typically, FPGA kernel compilation process takes few hours to compile the bit stream. Therefore, all kernels are emulated to verify the kernel performance and functionality on a host running 4-core Intel core I7-4810MQ 2.8 GHz processor and 64-bit CentOS 7.3 operating system. After verifying kernel functionality, all BP designs are executed on a BittWare 520N-MXboard which is equipped with an Intel Stratix 10-MX2100FPGA. The Intel Stratix 10 FPGA consists of hardened floating-point units (FPUs) dedicated to perform floating-point operations. The Intel Quartus Pro/AOCL 19.3 is used to compile and generate the FPGA bit-stream and BittWare BSP (BIST 1.0-1) is used to program the FPGA.

A. *FPGA Performance metric*

Performance of an FPGA is evaluated using its resource utilization and kernel speed for a given design. FPGA resource utilization report shows the usage statistics of Adaptive Look-Up Tables (ALUTs), registers, logic utilization, DSP Blocks, and RAM Blocks for kernel design. However, the implemented FPGA Performance Metric (FPM) only uses logic utilization and kernel F_{MAX} [38]. Generally, FPGA devices consist of thousands of Logic Array Blocks (LABs) that are connected to interconnects. In the Stratix 10 FPGA, each of these LABs contain ten Adaptive Logic Modules (ALMs) and each ALM contains a combinational Look-Up Table (LUT), two adders, and four registers [41]. Number of ALUTs and registers in the resource utilization report represents the number of half-ALMs used for the design and the number of logic utilization indicates the total number of ALMs necessary to implement the design [42]. Therefore, amount of logic utilization is a good indication to measure the FPGA resource usage.

Sometimes higher logic utilization leads to high kernel F_{MAX} but it highly depends on the compiler. Therefore, FPM metric is designed to generate higher values for designs that has high kernel speed and low logic utilization. The FPM value is a good metric to compare FlBP and FxBP designs fairly and it is given by,

$$FPM = \frac{f_{MAX}}{LU}, \tag{29}$$

where, f_{MAX} is the kernel speed (Hz) and LU is the logic utilization (logic elements).

V. Results

All BP designs are tested with synthetic phase history data obtained from the SAR simulator [26] for 512×512 sized images and real SAR data obtained from the GOTCHA dataset [39]. Then, resource utilization of each kernel design is obtained from the `ACL Quartus Report` and the back projected output image quality is evaluated by computing the peak signal to noise ratio (PSNR) of the output image.

A. *Resource Utilization and Kernel F_{MAX}*

The resource utilization is divided into five types, ALUTs, Registers, Logic Utilization, RAM Blocks, and DSP Blocks. These five resource types and kernel F_{MAX} (MHz) are considered individually to understand the resource utilization and performance of each design. Table II shows the resource utilization, kernel F_{MAX} for all BP kernel designs. The resource utilization in the `ACL Quartus Report` consists of kernel resource utilization and the BSP resource utilization. Therefore, resource utilization of `hello world` kernel is subtracted from the resource utilization of individual BP kernels to create a reference point and remove BSP resource utilization. All results shown in Table II are based on a reference design.

Table II: Resource Utilization and kernel F_{MAX} (MHz) of BP kernels

Resource	**Kernel Designs**			
Type	**FlBP-NDR**	**FxBP-NDR**	**FlBP-SWI**	**FxBP-SWI**
ALUTs	29634	**18530**	30426	**17020**
Registers	67097	**55849**	69171	**52686**
Logic Utilization	25820	**21464**	36687	**21081**
RAM Blocks	**233**	242	**220**	224
DSP Blocks	**69**	85	**72**	82
Kernel F_{MAX}	**405.67**	397.61	376.22	**408.32**

Table II shows that the FxBP designs have lower resource utilization of resource types ALUTs, Registers, and Logic Utilization. However, FlBP designs use less DSP Blocks and RAM Blocks compared to FxBP designs. This is mainly due to the square-root operation that uses 32-bit `float` type variables (Listings 1 and 2 lines 13-14) in floating-point based designs and 64-bit `long` type variables in fixed-point based designs. Since FxBP designs mostly use λ_R scaled up fixed-point values, all variables in square-root operation use 64-bit `long` type to avoid the overflow.

Further analysis of Table II shows that the logic utilization of the FxBP-NDR is less compared to FlBP-NDR but it has a lower F_{MAX} and generates the BP image output faster. This is another instance to showcase the importance of a performance metric for FPGA designs, since it is hard to evaluate kernels design solely on logic utilization or kernel speed. Table III shows the resources saved by FxBP designs compared to FlBP designs. It is evident that even with hardened FPUs, FxBP designs use less hardware resources than FlBP designs. Overall, FxBP designs use significantly fewer resources than FlBP designs.

B. *Performance Metric Evaluation*

FPGA performance is evaluated by calculating FPM value for each design. Table IV shows the logic utilization, F_{MAX}, and FPM for each design.

As shown in TableIV, FPGA performance is higher for FxBP designs compared to FlBP designs. The FxBP-NDR shows a 17.90% performance increase than FlBP-NDR and

Table III: Resource Savings from fixed-point based designs

Resource Type	**Kernel Designs**	
	FxBP-NDR	**FxBP-SWI**
ALUTs	37.47%	44.06%
Registers	16.76%	23.83%
Logic Utilization	16.87%	42.54%
RAM Blocks	-3.86%	-1.82%
DSP Blocks	-23.19%	-13.89%

Table IV: FPGA performance of all BP designs

	Kernel Designs			
	FlBP-NDR	**FxBP-NDR**	**FlBP-SWI**	**FxBP-SWI**
Logic Utilization	25820	**21464**	36687	**21081**
Kernel F_{MAX}	**405.67**	397.61	376.22	**414.25**
FPM	15711.46	**18524.51**	10254.86	**19650.40**
% improved FPM		**17.90%**		**91.62%**

the FxBP-SWI shows a performance increase of 91.62% than FlBP-SWI. Analyzing logic utilization data shown in Table III and FPGA performance shown in Table IV, it is clear that FxBP-SWI design is the best design choice compared to all other designs.

C. Back Projected Image Quality Comparison

All images shown in Figure 5 are processed with the SAR simulator [26] to create synthetic phase history for each image. Then the synthetic phase history data is given as an input to the BP modules to generate the back projected image output. Usage of synthetic phase history data is beneficial to evaluate performance of various BP design techniques, and objective and subjective image quality evaluation is possible by comparing the original image with the back projected image. Back projected output image quality is evaluated objectively by calculating PSNR values for each image output. All calculated PSNR (dB) values for all BP designs are shown in Table V. The PSNR values shown in Table V are categorized as FlBP and FxBP designs, since both NDR and SWI design techniques generate the same image output with the same PSNR value.

Table V: PSNR (dB) results for test images

Test images	**FlBP Designs**	**FxBP Designs**	**Error (Δ)**
Pentagon	25.2305	25.1438	0.0867
Airport	20.5262	20.4465	0.0797
San Diego	23.5844	23.4785	0.1059
Stockton	27.2434	27.0831	0.1603
Wash-ir	19.1631	19.2551	-0.0920
Average Error			**0.0681**

Note that the difference between PSNR values (Δ) of FlBP and FxBP designs are very low. On average, for all images, PSNR value is decreased by 0.0681 dB which is negligible. Therefore, FlBP and FxBP designs are generating very similar back projected images. Figures 6 and 7 show the back projected outputs from FlBP and FxBP designs. Figure 8 shows the back projected output of FlBP and FxBP designs for the 'parking lot' SAR data from the GOTCHA dataset [39]. As expected both designs generate image outputs for real SAR data that are visually indistinguishable.

VI. Conclusion

This paper introduces a new fixed-point based FPGA design for SAR back projection algorithm (FxBP) and shows the significance of fixed-point based designs for computationally complex algorithms. The proposed FxBP algorithm is designed with NDR and SWI kernel structures. The floating-point based back projection algorithm (FlBP) is designed with NDR and SWI kernel structures to compare with the FxBP designs. All designs are programmed and tested with a Stratix 10 FPGA device. It is shown that the FxBP designs save a significant amount of resource utilization compared to FlBP designs. Compared to FlBP designs, FxBP-NDR and FxBP-SWI designs save 16.87% and 42.54% logic resources, respectively. The FPGA performance metric shows that the FxBP-NDR and FxBP-SWI designs gain a 17.90% and 91.62% performance increase compared to FlBP designs. Also, the FxBP designs generate imagery of comparable image quality when compared to FlBP designs. Image quality is evaluated using PSNR and for all images the average PSNR decrease is 0.0681 dB. It is shown that for modern FPGA devices (with hardened FPUs), fixed-point based designs are still a significant option for computationally expensive algorithms and due to significant logic reduction it offers more flexibility to select devices for specific applications.

References

[1] Y. S. Rao, *Synthetic Aperture Radar (SAR) Interferometry for Glacier Movement Studies*. Dordrecht: Springer Netherlands, 2011, pp. 1133–1142.

[2] Y. Ban, P. Zhang, A. Nascetti, A. R. Bevington, and M. A. Wulder, "Near real-time wildfire progression monitoring with sentinel-1 sar time series and deep learning," *Nature Scientific Reports*, vol. 10, no. 1, p. 1322, 2020.

[3] M. Watanabe, C. N. Koyama, M. Hayashi, I. Nagatani, and M. Shimada, "Early-stage deforestation detection in the tropics with l-band sar," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 6, pp. 2127–2133, 2018.

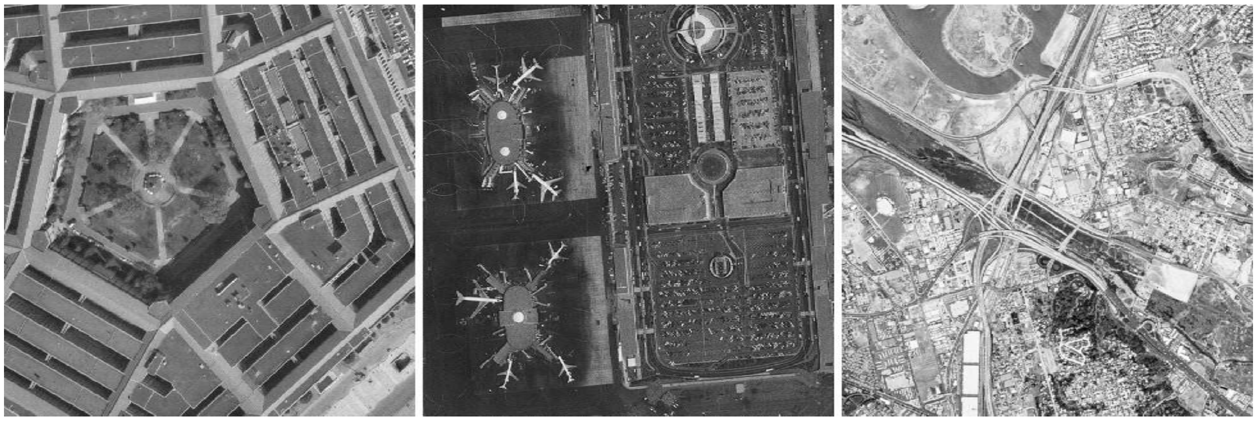
[4] F. M. Henderson and Zong-Guo Xia, "Sar applications in human settlement detection, population estimation and urban land use pattern analysis: a status report," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 35, no. 1, pp. 79–85, 1997.

[5] Y. Yamaguchi, "Disaster monitoring by fully polarimetric sar data acquired with alos-palsar," *Proceedings of the IEEE*, vol. 100, no. 10, pp. 2851–2860, 2012.

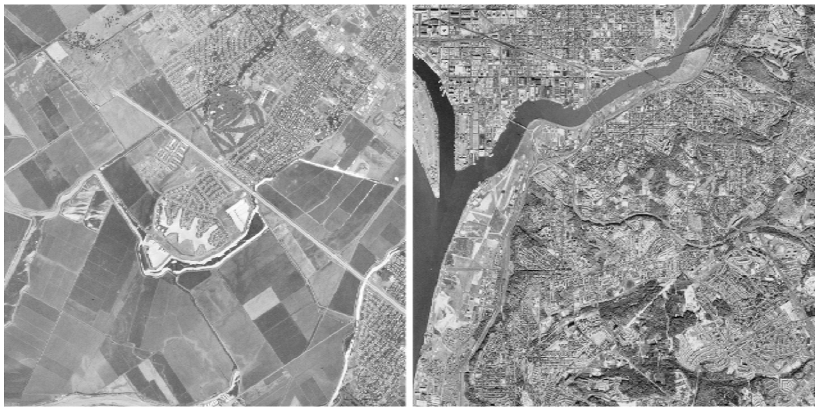
[6] H. L. Li, J. Li, Y. X. Hou, L. Zhang, M. D. Xing, and Z. Bao, "Synthetic aperture radar processing using a novel implementation of fast factorized back-projection," in *IET International Radar Conference 2013*, 2013, pp. 1–6.

[7] W. M. Brown and R. J. Fredricks, "Range-doppler imaging with motion through resolution cells," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-5, no. 1, pp. 98–102, 1969.

[8] R. Bamler, "A comparison of range-doppler and wavenumber domain sar focusing algorithms," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 30, no. 4, pp. 706–713, 1992.



(a) Left: Pentagon image, center: Airport image, right: San Diego image



(b) Left: Stockton image, right: Wash-ir image

Figure 5: Test images used to create synthetic video phase history (VPH)

[9] J. Mittermayer, A. Moreira, and O. Loffeld, "Spotlight sar data processing using the frequency scaling algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 5, pp. 2198–2214, 1999.

[10] D. An, Y. Li, X. Huang, X. Li, and Z. Zhou, "Performance evaluation of frequency-domain algorithms for chirped low frequency uwb sar data processing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 2, pp. 678–690, 2014.

[11] L. Zhang, J. Sheng, M. Xing, Z. Qiao, T. Xiong, and Z. Bao, "Wavenumber-domain autofocusing for highly squinted uav sar imagery," *IEEE Sensors Journal*, vol. 12, no. 5, pp. 1574–1588, 2012.

[12] H. Shin and J. Lim, "Omega-k algorithm for airborne forward-looking bistatic spotlight sar imaging," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 2, pp. 312–316, 2009.

[13] B. Liu, T. Wang, Q. Wu, and Z. Bao, "Bistatic sar data focusing using an omega-k algorithm based on method of series reversion," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 8, pp. 2899–2912, 2009.

[14] C. Cafforio, C. Prati, and F. Rocca, "Sar data focusing using seismic migration techniques," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 27, no. 2, pp. 194–207, 1991.

[15] R. K. Raney, H. Runge, R. Bamler, I. G. Cumming, and F. H. Wong, "Precision sar processing using chirp scaling," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 32, no. 4, pp. 786–799, 1994.

[16] F. H. Wong, I. G. Cumming, and Y. L. Neo, "Focusing bistatic sar data using the nonlinear chirp scaling algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 9, pp. 2493–2505, 2008.

[17] F. Li, S. Li, and Y. Zhao, "Focusing azimuth-invariant bistatic sar data with chirp scaling," *IEEE Geoscience and Remote Sensing Letters*, vol. 5, no. 3, pp. 484–486, 2008.

[18] A. F. Yegulalp, "Fast backprojection algorithm for synthetic aperture radar," in *Proceedings of the 1999 IEEE Radar Conference. Radar into the Next Millennium (Cat. No.99CH36249)*, 1999, pp. 60–65.

[19] Shu Xiao, D. C. Munson, S. Basu, and Y. Bresler, "An n2logn backprojection algorithm for sar image formation," in *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers (Cat. No.00CH37154)*, vol. 1, 2000, pp. 3–7 vol.1.

[20] L. A. Gorham and L. J. Moore, "SAR image formation toolbox for MATLAB," in *Algorithms for Synthetic Aperture Radar Imagery XVII*, E. G. Zelnio and F. D. Garber, Eds., vol. 7699, International Society for Optics and Photonics. SPIE, 2010, pp. 46 – 58. [Online]. Available: https://doi.org/10.1117/12.855375

[21] A. Sommer and J. Ostermann, "Explicit motion compensation for backprojection in spotlight sar," in *2016 17th International Radar Symposium (IRS)*, 2016, pp. 1–4.

[22] L. Chen, D. An, and X. Huang, "A backprojection-based imaging for circular synthetic aperture radar," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 8, pp. 3547–3555, Aug 2017.

[23] L. M. H. Ulander, H. Hellsten, and G. Stenstrom, "Synthetic-aperture radar processing using fast factorized back-projection," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 3, pp. 760–776, 2003.

[24] L. Zhang, H. Li, Z. Qiao, and Z. Xu, "A fast bp algorithm with wavenumber spectrum fusion for high-resolution spotlight sar imaging," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 9, pp. 1460–1464, 2014.

[25] D. L. N. Hettiarachchi and E. Balster, "An accelerated sar back projection algorithm using integer arithmetic," in *2018 Asia-Pacific Signal*

and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2018, pp. 80–88.

[26] E. J. Balster, F. A. Scarpino, A. M. Kordik, and K. L. Hill, "A simulator for spotlight sar image formation," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 2017, pp. 1–5.

[27] A. Fasih and T. Hartley, "Gpu-accelerated synthetic aperture radar backprojection in cuda," in *2010 IEEE Radar Conference*, 2010, pp. 1408–1413.

[28] E. J. Balster, M. P. Hoffman, J. P. Skeans, and D. Fan, "Gpgpu acceleration using opencl for a spotlight sar simulator," in *Proceedings of the 5th International Workshop on OpenCL*, ser. IWOCL 2017. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3078155.3078157

[29] B. Ge, L. Chen, D. An, and Z. Zhou, "Gpu-based ffbp algorithm for high-resolution spotlight sar imaging," in *2017 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, 2017, pp. 1–5.

[30] M. Wielage, F. Cholewa, C. Fahnemann, P. Pirsch, and H. Blume, "High performance and low power architectures: Gpu vs. fpga for fast factorized backprojection," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, 2017, pp. 351–357.

[31] D. Pritsker, "Efficient global back-projection on an fpga," in *2015 IEEE Radar Conference (RadarCon)*, 2015, pp. 0204–0209.

[32] F. Cholewa, M. Wielage, P. Pirsch, and H. Blume, "Synthetic aperture radar with fast factorized backprojection: A scalable, platform independent architecture for exhaustive fpga resource utilization," in *International Conference on Radar Systems (Radar 2017)*, 2017, pp. 1–6.

[33] M. W. Numan, B. J. Phillips, G. S. Puddy, and K. Falkner, "Towards automatic high-level code deployment on reconfigurable platforms: A survey of high-level synthesis tools and toolchains," *IEEE Access*, vol. 8, pp. 174 692–174 722, 2020.

[34] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[35] *Intel FPGA SDK for OpenCL pro edition: Best practices guide*, Intel Corporation, may 2019.

[36] M. Parker, "High-performance floating-point implementation using fpgas," in *MILCOM 2009 - 2009 IEEE Military Communications Conference*, 2009, pp. 1–5.

[37] *The industry's first floating-point FPGA*, Intel Corporation, 2014.

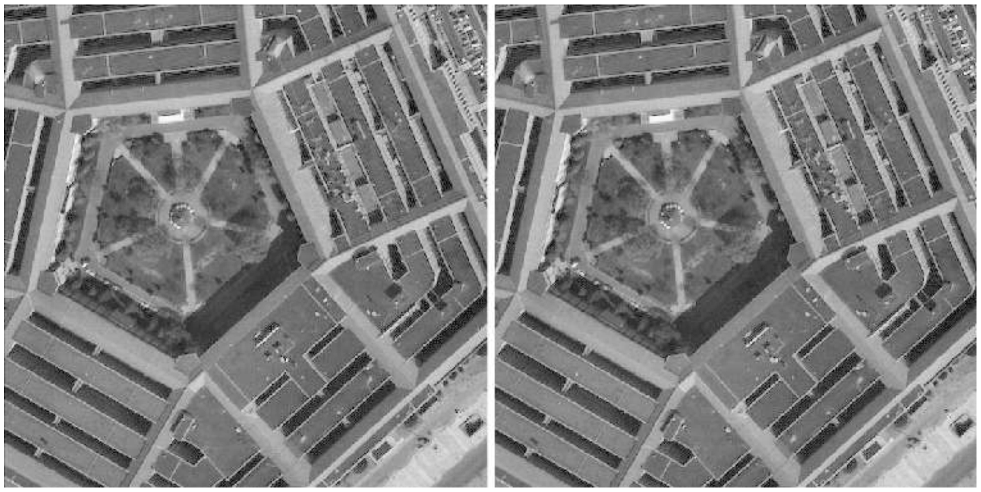
[38] D. L. N. Hettiarachchi, V. S. P. Davuluru, and E. J. Balster, "Integer vs. floating-point processing on modern fpga technology," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, pp. 0606–0612.

[39] Gotcha volumetric sar data set. [Online]. Available: https://www.sdms.afrl.af.mil/index.php?collection=gotcha

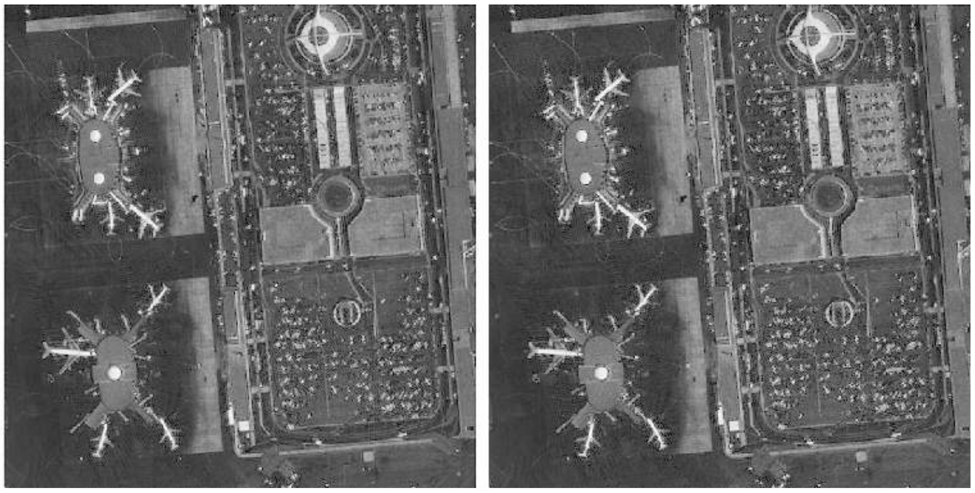
[40] J. C. French and E. J. Balster, "A fast and accurate orthorectification algorithm of aerial imagery using integer arithmetic," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 5, pp. 1826–1834, 2014.

[41] *Intel Stratix 10 Logic Array Blocks and Adaptive Logic Modules User Guide*, Intel Corporation, apr 2020.

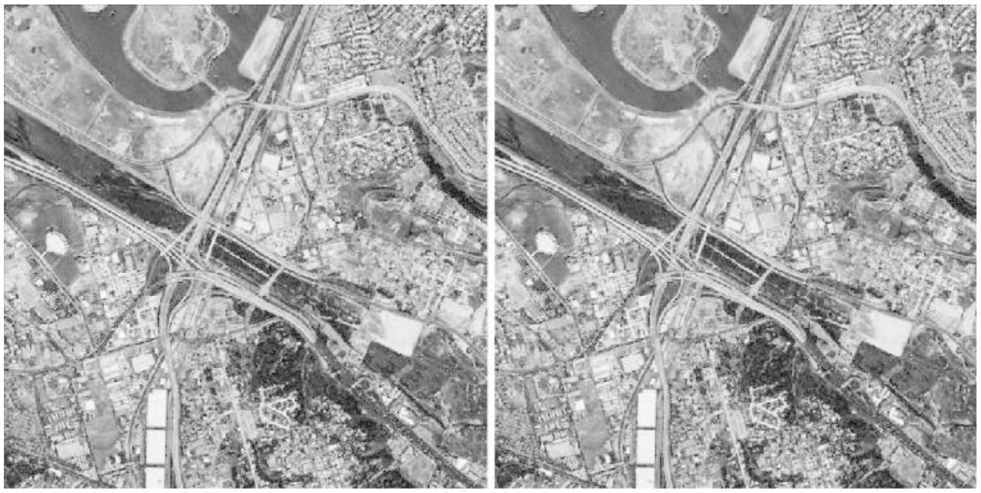
[42] *Fitter Resource Usage Summary Report*, Intel Corporation, 2017.



(a) Pentagon image

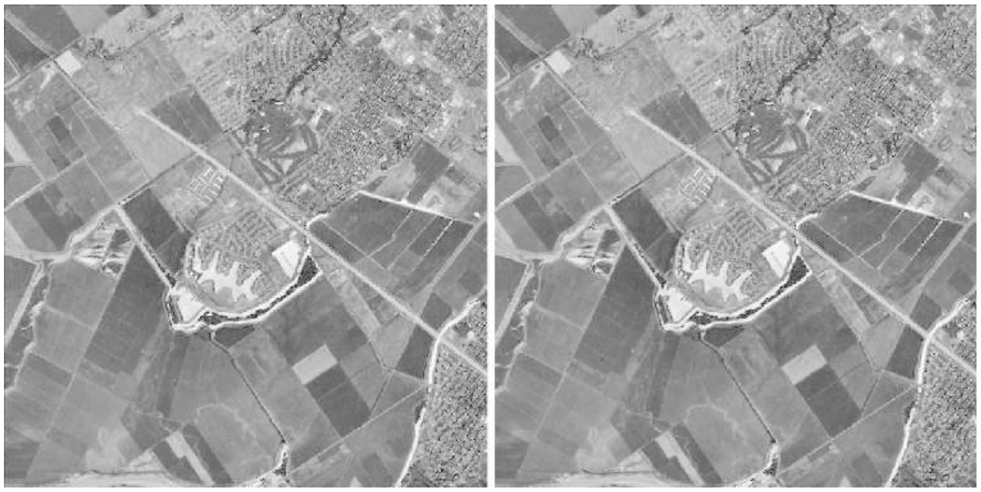


(b) Airport image

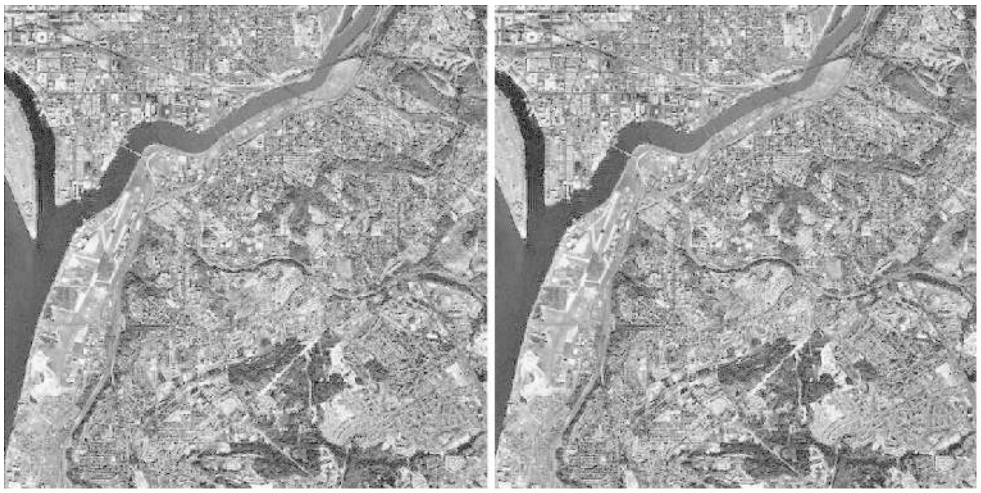


(c) San Diego image

Figure 6: Output image comparison. Left: FlBP design output, Right: FxBP design output

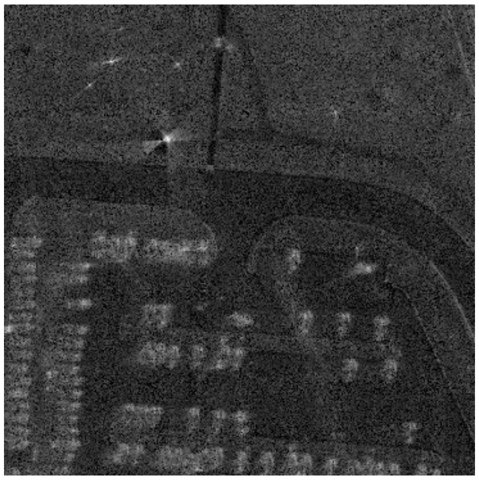


(a) Stockton image

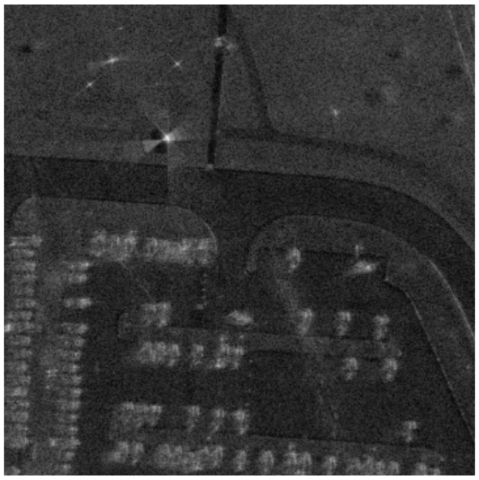


(b) Wash-ir image

Figure 7: Output image comparison. Left: FlBP desgin output, Right: FxBP design output



(a) Left: FlBP design output



(b) Left: FxBP design output

Figure 8: Back projected ‘parking lot’ image from GOTCHA dataset [39]