Synthetic Aperture Radar Back-Projection using Adaptive Intelligent Engines in AMD Versal™ Adaptive SoCs

Mark Rollins
Enbedded Business Group,
Advanced Micro Devices
Ottawa, ON, Canada
mark.rollins@amd.com

Ken O'Neill
Embedded Business Group,
Advanced Micro Devices
San Jose, CA, USA
koneill@amd.com

***Abstract*— Synthetic Aperture Radar (SAR) [1] creates a 2D or 3D image of stationary objects or landscapes from a moving platform such as an airplane or spacecraft. The distance travelled over the target by the platform creates a large synthetic antenna aperture that mimics a much larger antenna array yielding superior image resolution. Signal processing combines coherently many radar pulses collected from multiple platform positions above the target. Many algorithms exist to perform SAR. The Back-Projection (BP) algorithm for SAR is one of the easiest to understand. While the computational cost is high, BP lends itself naturally to parallel processing [2] and finds use in practical systems. This paper describes a reference design for BP-based SAR on Very Long Instruction Word (VLIW) vector processors in the form of the Adaptive Intelligent Engines (AI Engines, AIE) embedded in the AMD Versal XQRVC1902 adaptive SoC, which is qualified for space flight. Using the GOTCHA data set [1] with 586 radar pulses, the SAR engine achieves ~2.5 frames per second for a 512×512 image with fewer than 32 AIE tiles. With eight instances of the BP engine, the design achieves close to 20 frames per second using around 217 AIE tiles in an AMD Versal VC1902 adaptive SoC.**

The BP algorithm contains a half-dozen workloads with different characteristics that are combined together into an efficient data flow. The "GOTCHA Volumetric SAR Data Set" was captured by the U.S. Air Force Sensor Data Management System and is made available for public download. The repository consists of SAR phase history data collected at X-band with a 640 MHz bandwidth with full azimuth coverage at 8 different elevation angles with full polarization. The target scene consists of many civilian vehicles and calibration targets.

Keywords—SAR, radar, Versal, Adaptive SoC

I. System Modeling

We describe the development of the MATLAB system model of the SAR Back-Projection algorithm implemented on the Versal adaptive SoC AI Engines. The purpose of the system model is three-fold:

1. Capture the ideal algorithm model and evaluate its baseline system performance using the GOTCHA data set.

2. Identify the compute workloads required for the AI Engine implementation.

3. Model any algorithmic imperfections induced by the AI Engine solution and assess their performance impact.

We start with the baseline MATLAB model for SAR published by Gorham & Moore in [2]. This model is well-known and accepted in the SAR community and provides a solid baseline, and also uses the GOTCHA data set for comparison purposes..

A. Inner Loop Analysis

The inner loop of the system level MATLAB model as outlined in [2] is shown in the following block code:

```
Line  1:  data_o.r_vec = linspace(-data_o.Nfft/2,data_o.Nfft/2-1,
            data_o.Nfft)*data_o.maxWr/data_o.Nfft;
Line  2:  % Loop through every pulse:
Line  3:    for ii = 1 :  data_o.Np
Line  4:      % Form the range profile with zero padding added:
Line  5:      rc = fftshift(ifft(data_o.phdata(:,ii),data_o.Nfft));
Line  6:      % Calculate differential range for each pixel in the image (m):
Line  7:      dist_sq = (data_o.AntX(ii)-data_o.x_mat).^2 + ...
Line  8:                (data_o.AntY(ii)-data_o.y_mat).^2 + ...
Line  9:                (data_o.AntZ(ii)-data_o.z_mat).^2;
Line 10:      dR = sqrt(dist_sq) - data_o.R0(ii);
Line 11:      % Calculate phase correction for image:
Line 12:      phCorr = exp(1i*4*pi*data_o.minF(ii)/c*dR);
Line 13:      % Determine which pixels fall within the range swath:
Line 14:      I = find(and(dR > min(data_o.r_vec), dR < max(data_o.r_vec)));
Line 15:      % Update the image using linear interpolation:
Line 16:      dist_part = interp1(data_o.r_vec,rc,dR(I),'linear');
Line 17:      data_o.im_final(I) = data_o.im_final(I) + dist_part .* phCorr(I);
Line 18:    end % ii
```

This captures the full algorithmic processing for a single radar pulse. The final SAR image is obtained by processing a large number of radar pulses and combining them coherently. The MATLAB model of the SAR BP algorithm when run on the GOTCHA data set for the 40 degree azimuth scenario outlined above yields an output image shown in the paper. This matches very well to the reference image shown in [2].

B. SAR Back Projection Compute Workloads

Analysis of the MATLAB code helps to identify the various compute workloads for the SAR BP algorithm as follows:

• Line 5: The algorithm requires an IFFT that is applied to the radar phase pulse data. A single transform is computed per radar pulse. Its output is used to update all pixels in the target image.

• Line 7: A squared distance measure is computed between the antenna platform (AX,AY,AZ) and every position (x,y,z) in the target scene.
• Line 10: The differential distance dR is computed as the difference between the target distance and the range to the scene center R0. This requires a sqrt() and subtraction operation for every pixel in the target image.
• Line 12: A complex-valued phase correction term is computed as the output of a complex exp() function. Its input argument is the differential distance dR scaled by a number of the radar parameters, π, Fmin and c, the speed of light. Once again, these computations must be performed for every pixel in the target image.
• Line 14: This indexing check ensures no computed distances fall outside the radar range. This may be avoided in practice by careful dimensioning of the solution parameters.
• Line 16: A one-dimensional linear interpolation is made between the differential distance dR based on the radar range profile rc computed by the IFFT.
• Line 17: Finally, each pixel in the target image is updated by adding the complex-valued product of the phase correction term and the interpolated differential distance.

Based on the analysis above, the following AI Engine kernels are identified to service the overall set of compute workloads:
• ifft() -- implement the IFFT transform
• diff3dsq() -- compute the 3D squared distance
• sqrt() -- compute the square root of the squared distance
• dR_comp() -- compute the difference between target distance and the range to scene center
• fmod_floor() -- reduce the input argument to exp() modulo 2π (may not be obvious without the discussion below)
• expjx() -- compute the complex exponential function
• interp1() -- interpolate the differential distance against the fixed radar grid
• bp_update() -- update the SAR target image using the phase correction and distance terms.
Note that all AI Engine kernels will use single-precision floating-point data types, as they then may target the vectorized floating-point data path for the AIE architecture to yield good performance.

Early prototyping work validates the performance of these algorithm variants in the context of the MATLAB system model using Vitis Functional Simulation. These results are summarized in a functional block diagram of the computation engine for SAR BP implemented using AI Engines, shown in Fig. 1 below.

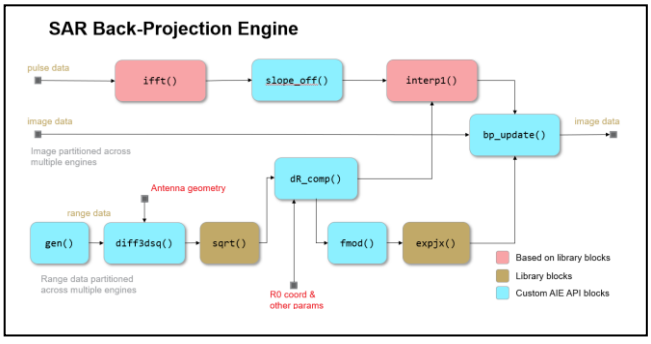

Fig. 1. SAR back projection engine block diagram

II. System Partitioning

The performance of this new BP algorithm was evaluated using the system model context with AMD Vitis Functional Simulation of some early AI Engine implementation models. Having confirmed the system performance is acceptable, we then describe the system partitioning work to identify a feasible architecture, data flow, and kernel partitioning that leads to a workable design with attractive performance characteristics and cost effective resource profile.

A. System Parameters and Performance Targets

A first system partitioning step identifies the system parameters to which the SAR BP engine will be designed, shown in Table I below. These are influenced strongly by the GOTCHA data set proposed for evaluating the system performance. These parameters drive the overall system performance and cost of the solution. The computational complexity of the algorithm is $O(N^3)$ for N×N pixel images. The workload scales linearly with the number of radar pulses to be combined coherently. The IFFT cost varies as (N·log (N)) and can require a large memory footprint.

TABLE I. SAR System Parameters

Parameter	Value	Notes
Image Width × Height	512 × 512 Pixels	Square image
Number of Pulses	586 Pulses	For 5 azimuth angles
Target Throughput	1 GOP/sec	Rate of per-pixel back projection operations
IFFT Transform Size	2048 Points	Based on system model
Final Frame Rate	6.5 fps	All pulses accumulated
Per-Pulse Frame Rate	3820 fps	For a single pulse
Image Storage in DDR	2 MB	Assume 8B per pixel
IFFT Transform rate	3820 Hz	One transform per radar pulse
IFFT Sampling rate	8 Msps	Assume streaming solution
Total Number of AI Engines	TBD	Requires prototyping

B. AI Engine Prototyping

All required compute workloads were identified during system modeling and validated in terms of their algorithmic performance, and must be quantified for their throughput performance and resource requirements. Results of prototyping are listed here and summarized in Table II below.

• A library block was used to prototype ifft4k() quickly. This requires 11 tiles. It's likely the smaller ifft2k() design will require half as many tiles, so including this larger design will build margin into our resource estimate.
• No prototype was build for the dR_comp() kernel as it represented a simple subtraction workload that should yield excellent performance. A guess of 3 tiles resources is used to align with the larger memory footprint found in the other prototyped kernels. This introduces some additional margin to reduce risk.
• The sqrt_lib(), cos_lib() and sin_lib() kernels use library blocks for quick prototyping.

• The interp1() kernel was not prototyped because it is anticipated to use the underlying library block implementation with some custom coding to manage asynchronous buffers. Here it's performance and resources are based on the library blocks.

TABLE II. BP Engine Resource Requirements

Kernel	Throughput (Mbps)	AIE Tiles Used	Notes
ifft4k()	180	11	Trim to 5 tiles for 2K-pt
diff3dsq()	433	2	
dR_comp()	no proto	3	Estimate # tiles
sqrt_lib()	420	2	Library
fmod_floor()	375	1	Need to optimize
cos_lib()	420	3	Library
sin_lib()	420	3	Library
interp1()	420	4	Estimate based on Library
bp_update()	620	2	
Overall	**400**	**31**	**Target 4 x 8 array**

C. *Projected System Throuput*

Given the proposed architecture of a single SAR BP engine as a 4×8 rectangular array of tiles capable of achieving ~400 Msps pixel-by-pixel throughput, these assumptions can be used to investigate the capability of larger solutions using several identical instances of this baseline engine. Each engine instance could be assigned its own portion of the target output image to realize a linear scaling in throughput capacity. The baseline engine achieves a system frame rate of 2.6 fps. A design with 8 engine instances can achieve a system frame rate of 20.8 fps in principle.

D. *Back Projection for SAR on AI Engines*

The SAR BP engine is built using Versal AI Engines. A single BP engine instance provides a complete implementation of the SAR BP algorithm. Multiple instances of the engine may be used to increase throughput by partitioning a non-overlapping portion of the target image to each engine instance. This is considered later in the paper. All compute workloads are partitioned to the AI Engine array. Details of implementation are described in the paper. The graph view for the SAR BP engine is shown in Fig. 2.

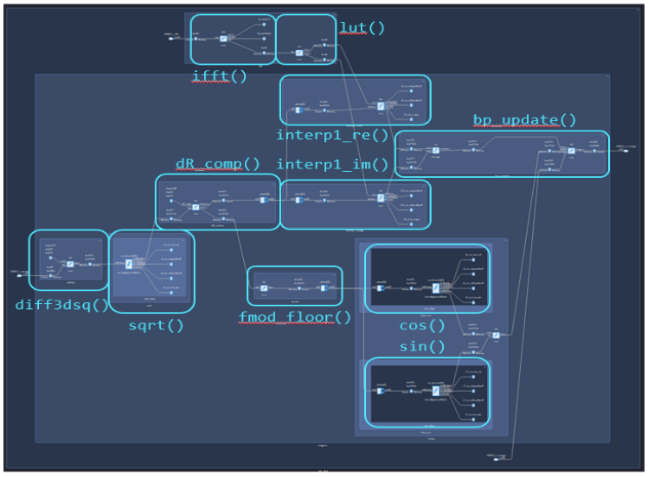

Fig.2. Graph View of SAR BP Engine using AI Engines

E. *Resource Utilization, Throughput and Latency*

The AI Engine resources for the SAR BP engine are shown in Table III below. The design uses 14 tiles for compute and 26 tiles overall for compute and buffering. Throughput and latency are assessed. An approximate frame rate is calculated to be 2.6 frames per second, which is very close to the frame rate predicted during system partitioning. The latency of the design can be considered as the time required for processing the full target image less the upload time assuming the image is produced as new IFFT radar pulses are streamed into the engine. In this case, the latency includes that of the IFFT processing plus the time required to process all the radar pulses. This amounts to approximately 390ms for a full complement of 586 radar pulses.

TABLE III. AI Engine Resource Utilization
(Single Engine Design)

Tiles used for Kernels, Buffers or Nets	30 of 400 (7.5%)
Tiles used for AI Engine Kernels	14 of 400 (3.5%)
Tiles used for Buffers	26 of 400 (6.5%)
Tiles used for Stream Interconnect	17 of 400 (4.3%)
GMIO Input Channel Usage	1 of 32
GMIO Output Channel usage	0 of 32
PLIO Input Channel Usage	1 of 312
PLIO Output Channel Usage	1 of 234
DMA FIFO Buffers	0
Interface Channels used for ADF Input / Output	3
Interface Channels used for Trace Data	0

F. *Hardware*

The single engine version of the design is run in hardware on an AMD VCK190 evaluation board. Each of the AI Engine kernels shown in the system block diagram are assessed for performance and device utilization.

G. *Final SAR BP Engine Performance*

Figures 3 and 4 below compare the performance of the final AI Engine implementation with the ideal MATLAB baseline when run on the GOTCHA data set with a total of 586 radar pulses. This run illustrates the final performance with all the implementation artifacts. Quality of results indicated by the Structural Similarity Index Measure (SSIM) [3] metric has a value of 0.9965 and the Peak Signal to Noise (PSNR) has a

value of 74.1 dB. These compare favorably to the early results computed during system modeling.

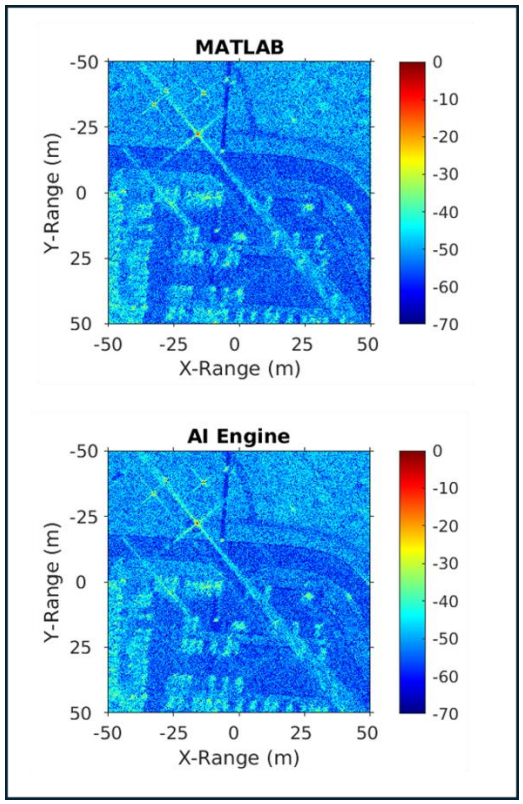

Fig. 3. Image results compared to MATLAB model

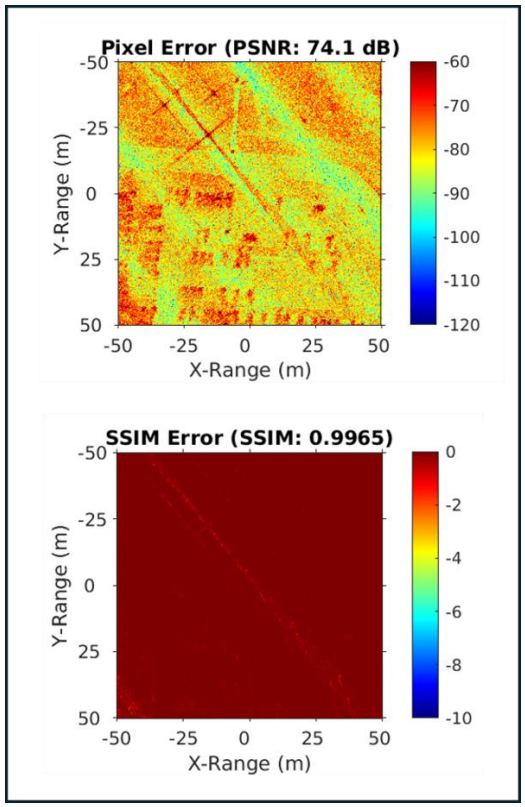

Fig. 4. Quality of results compared to MATLAB model

III. Multiple Engine Design and Higher Throuput

An 8-engine design has been developed by stepping and repeating the single engine design described above. It achieves approximately 8X throughput improvement by splitting the target SAR image into eight pieces, and instantiating eight separate engines to process each piece in parallel. The design is constructed with eight identical graphs. There is no separate top-level wrapper graph here. This makes the design extension quite easy but does consume more device resources than strictly required. The 8-engine design has been implemented in hardware on an AMD VCK190 evaluation board. The final throughput has been recorded at 19.3 frames per second. This is only slightly lower than 8X times the frame rate achieved with the single engine.

TABLE IV. AI Engine Resource Utilization (8-Engine Design)

Tiles used for Kernels, Buffers or Nets	193 of 400 (48.3%)
Tiles used for AI Engine Kernels	112 of 400 (28.0%)
Tiles used for Buffers	208 of 400 (52.0%)
Tiles used for Stream Interconnect	92 of 400 (23.0%)
GMIO Input Channel Usage	8 of 32
GMIO Output Channel usage	0 of 32
PLIO Input Channel Usage	8 of 312
PLIO Output Channel Usage	8 of 234
DMA FIFO Buffers	0
Interface Channels used for ADF Input / Output	24
Interface Channels used for Trace Data	0

A. Opportunities for Optimization

The 8-engine design was achieved quickly by instantiating the single engine design repeatedly. This creates a few opportunities for optimization:

• Each engine is using its own IFFT engine to transform the radar pulse. In reality, this operation is common to all engines and could be done with a single IFFT graph. The output of that common graph could then be broadcast to all 8 engines. This would save seven instances of 6 tiles or approximately 40 tiles. It will also remove 7 GMIOs from the design which will dramatically reduce the bandwidth required by the internal Network on Chip (NoC) to deliver the radar pulses to the AI Engine array from DDR.
• Constructing an 8-engine design with a single IFFT would require some code restructuring since routing the IFFT graph output to all engines would require a new top-level graph. This complicates the "Stamp & Repeat" approach to placement but would be manageable.
• The consumption of internal memory can in principle be reduced by partitioning the image buffers to external DDR memory instead of the internal memory, allowing the internal memory to be used for other functions. In this case, the radar processing would require eight GMIO pairs, one pair for each engine. The data flow would proceed from DDR, streaming the input image to each engine over the NoC to the AIE array, updating each image segment by its engine, then streaming the output image back to DDR over the NoC. This would remove all PL resources from the design -- a significant

savings and simplification. The DDR buffer design needs to be optimized to maximize the burst bandwidth available to each engine.

IV. Conclusion

This paper presents a detailed design of a Back-Projection engine for Synthetic Aperture Radar using VLIW vector processors in the AMD Versal adaptive SoC architecture. Using the GOTCHA data set with 586 radar pulses, the design achieves ~2.5 frames per second for a 512×512 target image with fewer than 32 AI Engine tiles. Eight instances of the engine achieve nearly an 8-fold increase in throughput.

Full device-level designs for both the single engine and 8-engine versions of the design are available as a reference for designers who are seeking to integrate on-orbit SAR back-projection processing into a single integrated circuit, achieving unprecedented levels of integration and reconfigurability for such an application. The paper also explores opportunities for further design optimization to more efficiently use the available programmable logic resources of the target adaptive SoC, and subsequently reduce power consumption.

References

[1] U.S. Air Force, "GOTCHA Volumetric SAR Data Set", U.S. Air Force Sensor Data Management System.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] L.A. Gorham & L.J. Moore, "SAR Image Formation Toolbox for MATLAB", SPIE Defense, Security, and Sensing, Orlando, FL, 2010.

[3] Wikipedia, "Structural Similarity Index Measure".

[4] AMD Tutorial on SAR Back Projection https://github.com/Xilinx/Vitis-Tutorials/tree/2025.1/AI_Engine_Development/AIE/Design_Tutorials/21-Back-Projection-SAR