FPGA Implementation of a Scalable SAR Image Processor for CFAR Object Detection

Dominik Günzel[a]
[a]German Aerospace Center (DLR), Maritime Safety and Security Lab, Am Fallturm 9, 28359 Bremen, Germany

Abstract

Ship detection on satellite SAR imagery regularly relies on CFAR algorithms for bright pixel discrimination. However, these algorithms are computationally complex, resulting in long processing times especially for high-resolution images, which is in conflict with the time-criticality of ship detection products. To satisfy the requirement of high product resolution and rapid delivery, an FPGA-based implementation of cell-averaging CFAR is presented. For the first time, High Bandwidth Memory is employed to eliminate previous memory bottlenecks and make full use of a pipelined datapath. Processing of a full Sentinel-1 IW high-resolution scene finishes in less than 8 seconds, while even higher resolution products such as TerraSAR-X StripMap are supported as well and processed in under 15 seconds. At the same time, power efficiency improves by an order of magnitude compared to a conventional server-grade CPU.

1 Introduction

Radar target detection relies on the reflective properties of certain scatterers resulting in high backscatter compared to their surroundings. A target is detected if the backscatter signal amplitude exceeds a certain threshold. However, the backscatter signal containing the desired echo is influenced by external clutter and interference as well as internal noise of the receiver. These undesired influences in combination with the non-static background lead to a variable rate of false alarms if a fixed threshold is applied to the backscatter signal. Constant False Alarm Rate (CFAR) algorithms are an established approach for improving radar target detection by applying an adaptive threshold [1]. By adapting the threshold to the local background statistics a desired false alarm rate can be maintained.

CFAR algorithms are widely used not only in radar systems processing range-cells in one-dimensional sliding windows, but also for processing Synthetic Aperture Radar (SAR) imagery. Here, a two-dimensional sliding window is applied to each target cell to detect pixels with high intensity caused by strong scatterers such as buildings, wind turbines, ships etc.

German Aerospace Center (DLR) has developed a ship detector for Maritime Situation Awareness that has been deployed and proven its operational usefulness in the ground station Neustrelitz [2]. The detector supports various SAR missions, such as TerraSAR-X and Sentinel-1, to help find and locate vessels on the oceans. Because locations of moving ships deprecate within minutes it is imperative that the time-sensitive information products are generated and delivered to the interested stakeholders as soon as possible after the SAR data is available. In this regard CFAR algorithms for ship detection pose a challenge as they are computationally demanding, especially when a two-dimensional sliding window is applied to high-resolution image data [3]. In order to minimize the processing latency impact and facilitate timely product delivery, powerful hardware with proportionally high energy consumption is currently deployed.

This paper contributes a novel implementation of a Cell-Averaging (CA) CFAR algorithm on Field Programmable Gate Array (FPGA) as a power-efficient attempt at achieving low latency even for high-resolution SAR imagery. The proposed detector is intended as a drop-in accelerator into existing ground-based processing infrastructure, which typically comprises traditional server hardware. FPGA designs on the other hand are often implemented on dedicated boards suitable for standalone deployment at the edge. Integration of FPGAs into data centers has historically been difficult as significant additional development effort needed to be invested into data and control interfacing between the server and FPGA. Nowadays, FPGA accelerator cards attempt to bridge this gap by leveraging the standard PCI express (PCIe) interface used by other extension cards such as Graphics Processing Units (GPUs), which are commonly deployed for machine learning tasks in High Performance Computing (HPC). The implemented CFAR detector primarily targets aforementioned FPGA accelerator cards in the data center, but may in principle be adapted to edge processing in future. State-of-the-art High Bandwidth Memory (HBM) is leveraged to feed data into a parallelized processing pipeline, while at the same time offering flexibility with regard to different SAR missions and product types. The following sections describe the implementation of the hardware processor and discuss the achieved results.

2 Related Work

Implementations of various types of CFAR algorithms on FPGAs have been published for traditional radar signal

processing, where a one-dimensional sliding window is applied along each range line to detect targets in real-time. For example, Cumplido et al. [1] proposed a design that can switch between CA-, Minimum and Maximum CFAR. They use a fixed configuration of 32 CFAR window background and 8 guard cells, which is common in radar target detection. However, this approach is unsuitable for processing two-dimensional SAR images from earth observation satellites. An FPGA-based implementation of a CFAR processor as part of an on-board ship detector for optical satellites was presented by Ghosh et al. [4]. As their design is targeted towards optical image data with medium resolution in the range of $50\,\mathrm{m}$ only very small sliding windows of up to 9×9 background pixels are supported. Despite this limitation, a long processing latency of $3.5\,\mu\mathrm{s}$ for each target pixel is reported. This is incompatible with SAR StripMap products such as TerraSAR-X single pol. RE with $3.25\,\mathrm{m}$ pixel spacing, where each scene can be up to 25000×40000 pixels large, as the processing latency would be over 58 minutes for a single scene.

In the frame of the EO-ALERT project, a CFAR detector for SAR satellite on-board processing was developed [3]. The FPGA-based implementation supports run-time configurable window sizes of up to 255×255 pixels. Results showed that data throughput of the used Double Data Rate (DDR) memory was a bottleneck, stalling the CFAR threshold calculation pipeline, so that the processing latency increased proportionally to the configured window size. While the CFAR threshold calculation pipeline serves as the basis of the improved detector for ground-based processing in this contribution, the memory interface was redesigned completely to support HBM, which offers much higher theoretical data throughput.

3 Algorithm Description

The ship detector is implemented using the CA-CFAR algorithm described by Frost et al. [5]. As illustrated in **Figure 1**, a background window is applied to each pixel on the SAR image. Because ships of relevant sizes usually comprise of more than one pixel an area of guard pixels around the potential target is excluded, resulting in the CFAR window frame marked in grey color. To obtain the statistics of this local background the intensities of the pixels therein are considered.

The intensities of background pixels over open water are assumed to follow a gamma or k distribution, but approximation with a Gaussian distribution has been shown to marginally impact the performance of object detection on SAR images, while significantly reducing the computational complexity of the background statistics [5]. Utilizing this simplification, the local reference threshold T for each pixel is calculated with the mean μ_b and standard deviation σ_b of the background pixel intensities as follows:

$$T = \mu_\mathrm{b} + k \cdot \sigma_\mathrm{b} \tag{1}$$

If the intensity of the pixel of interest exceeds the threshold, the pixel is denoted as a potential ship. The parameter k is referred to as the CFAR constant which scales the threshold and thus influences the false alarm rate. In operation at DLR, $k = 15$ has shown to provide a good balance between false alarm rate and probability of detection.

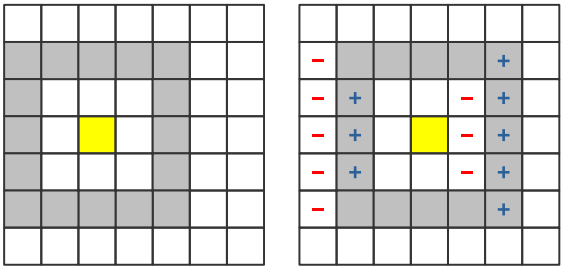

Figure 1 Illustration of CFAR background statistics calculation using sliding window. For each target pixel marked in yellow, all pixels in the grey background frame have to be considered excluding pixels in the guard area near the target (left). Computation can be optimized for consecutive targets in the same image row as some pixels remain in the background (right).

The mean of the background intensity μ_b is obtained by adding up the intensities of the included pixels and dividing by the number of pixels. For the standard deviation σ_b the intensities have to be squared individually before adding them up. Accumulating the background pixels in this way for each pixel on the image incurs a heavy computational load, especially for large size windows, because the complexity is $O(n^2)$. Fortunately, the algorithm can be optimized by saving the sum and squared sum of the current pixel under test for the next one. As is shown in **Figure 1**, only the first target pixel needs full accumulation because the background windows of neighboring pixels overlap. For consecutive targets in the same row only the pixels in the columns marked with $-$ have to be subtracted from the previous sums. Likewise, pixels in columns marked with $+$ have to be added to the sums. With this optimization complexity reduces to $O(n)$, which is beneficial especially for large background window sizes necessary when processing high-resolution SAR imagery.

4 Implementation

In this section, the implementation of the enhanced CFAR processor is explained. The target hardware is introduced (**Section 4.1**), followed by a more detailed look at the HBM interface (**Section 4.2**). As loading data into the cache for buffering CFAR window pixels presented a bottleneck in the previous design, the memory interface is completely redesigned (**Section 4.3**), which requires partitioning of the SAR image data in the HBM (**Section 4.4**).

4.1 Target Hardware

The CFAR detector targets data center accelerator cards to facilitate integration into existing ground station HPC infrastructure (see **Section 1**). For the sake of minimizing the significant development effort often associated with FPGA implementation, design re-use is a common approach. In

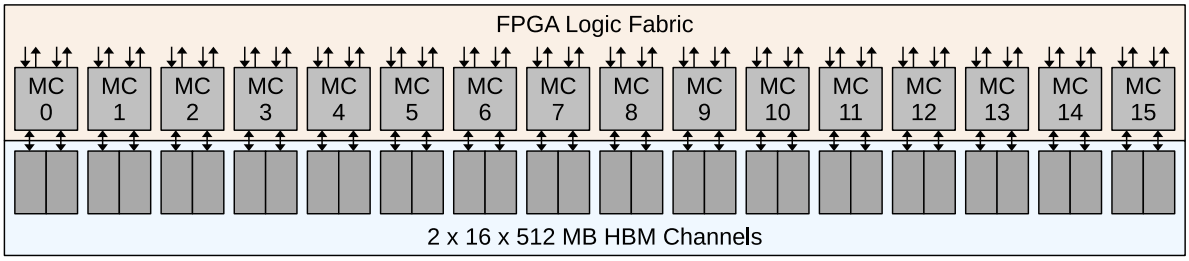

Figure 2 Simplified circuit diagram of the HBM architecture. 16 memory controllers residing in FPGA logic fabric are connected to pairs of the 32 HBM regions.

this case, the CFAR threshold calculation pipeline module from the EO-ALERT project mentioned in **Section 2** is re-used.

The design for satellite on-board processing was bottlenecked by the DDR memory where the image data was stored [3]. Therefore, the AMD Alveo u55c data center accelerator card was chosen as it is equipped with $16\,\mathrm{GB}$ of second-generation HBM with much higher theoretical throughput of up to $460\,\mathrm{GB/s}$ compared to the previous DDR4 with up to $19.2\,\mathrm{GB/s}$. The card features a Virtex XCU55 UltraScale+ FPGA manufactured in a $16\,\mathrm{nm}$ FinFET node, which promises high energy efficiency. With a standard PCIe x16 connector the u55c slots into most server mainboards. Interfacing with the Central Processing Unit (CPU) of the host server in this way provides low latency, high throughput data transfer between the host and accelerator card, which enables offloading of heavy computations from the general-purpose CPU to specialized hardware, in this case the FPGA.

4.2 High Bandwidth Memory

HBM aims to achieve substantially higher data throughput compared to DDR by bringing the Dynamic Random Access Memory (DRAM) memory chips close to the logic chip and packaging as a single unit. In the case of Alveo u55c, the HBM2 memory chips are connected directly to the FPGA fabric on a silicon interposer, which allows for a very wide memory bus. A simplified schematic of this interface is shown in **Figure 2**. The $16\,\mathrm{GB}$ memory is divided into $32 \times 512\,\mathrm{MB}$ so-called pseudo-channels that are connected pairwise to 16 memory controllers (MCs). These controllers translate from the physical DRAM interface to the Advanced eXtensible Interface (AXI), which is a common on-chip communication bus protocol. In short, there are 32 AXI interfaces with a port width of 256 bits each for reading and writing the memory. Because the $460\,\mathrm{GB/s}$ theoretical throughput of the HBM can only be reached if all read/write channels are utilized the distribution of data across the memory regions has to be considered (see **Section 4.4**). Compared to DDR with a single memory interface design complexity increases significantly.

4.3 Pixel Caching

To implement the CFAR threshold calculation in hardware, the pipelined processor presented in [3] was re-used and modified to support larger window sizes. A cache memory in front of the pipeline ensures optimal utilization by buffering the 16-bit pixel data read from the HBM. The pixel cache contains a number of parallel indexed queues that are implemented with Block RAM (BRAM) FPGA resources. Each queue has a capacity of $16\,\mathrm{KiB}$ or 8192 pixels. Columns of the CFAR window are stored across these memories, meaning that one row of the window is stored in BRAM 0, the row below is in BRAM 1 and so on. Therefore, the number of parallel memories defines the maximum CFAR window size. **Figure 3** shows the default configuration of 300 queues, but this number is configurable at compile-time using a single parameter, so that the design can be adapted to different missions and products. Furthermore, using fewer memory blocks results in a smaller circuit that may be integrated into an FPGA with less available resources. This will be revisited in **Section 5.1** as generating a smaller design yields higher performance for Sentinel-1 products specifically.

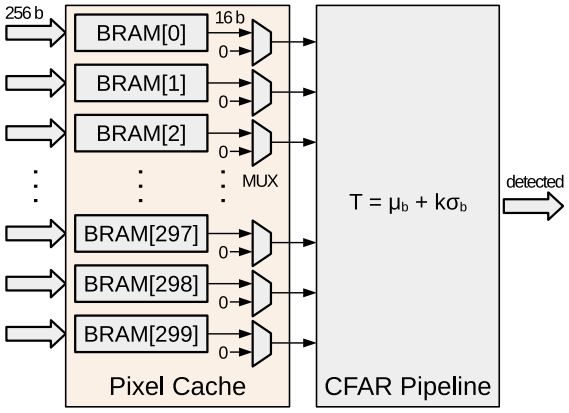

Figure 3 On-chip cache for buffering the CFAR window pixel data implemented with dual-port block RAM.

In the previous design, pixel data read from DDR-DRAM was written to the cache by unpacking the 512-bit data beats and multiplexing each 16-bit pixel to the target BRAM queue. Unpacking of the data beats is no longer necessary because the cache now supports asymmetric widths of the read and write sides. A full 256-bit HBM AXI beat containing 32 pixel intensities can be written to a queue per clock cycle, while the read interface is still 16 bits wide, i.e. the size of one pixel. The HBM with 32 AXI interfaces requires a more sophisticated multiplexing scheme compared to having a single DDR-DRAM interface. To avoid bus turnarounds, which impact throughput,

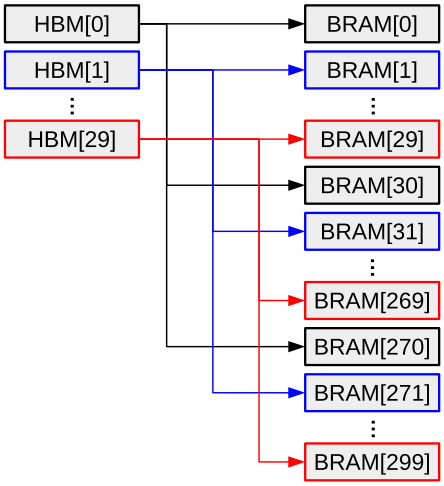

Figure 4 Connections from HBM read channels to block RAM write inputs.

only 30 AXI channels are assigned to reading while one channel is dedicated to writing the detection results back to memory. Since the results are represented by one bit per pixel, where 0 means the target is not detected and 1 means that it is, the data size is small and fits in a single 512 MB memory region. Thus, the last channel is unused because it shares the same HBM controller.

A logical extension of the DDR design would be to connect each of the 30 AXI read channels to all 300 BRAM queues. However, the physical routing required on the FPGA is practically impossible to realize. A more efficient solution is shown in **Figure 4**, where each HBM read channel is only connected to a subset of the BRAM inputs. HBM 0 is connected to BRAMs 0, 30,...270, while data from HBM 1 can only be written to BRAMs 1, 31,...271 etc.

4.4 Data Partitioning

Each HBM pseudo-channel can only access 512 MB of the total 16 GB capacity (see **Section 5.1**), but high-resolution 16-bit SAR images with large coverage regularly exceed 512 MB data size. Instead of storing the whole image in each HBM region, the data is distributed across the 30 read channels. Considering the HBM to BRAM multiplexing explained in the previous section, image data is split so that rows 0, 30, 60,... are stored in HBM region 0. Rows 1, 31, 61,... reside in region 1, and so on. This partitioning has a significant consequence for the pixel caching because it limits which BRAMs the image rows can be written to. For example, data of image row 0 can only be written to BRAMs 0, 30, ..., 270. Only if the row index of the topmost pixel in the window is 0, the topmost pixel is actually also stored in BRAM 0. For subsequent rows the window shifts downwards in the BRAM array, eventually wrapping around. This shift of the data in the cache requires careful housekeeping of where the background and guard window pixels are stored in each row. The multiplexers in **Figure 3** are used to mask the outputs of BRAMs holding guard pixels. In the next section it will be evaluated on the FPGA whether the extra implementation effort leads to higher throughput of the CFAR detector.

5 Integration and Results

Before the CFAR kernel can be deployed it has to be integrated on the target hardware. In this section, the hardware (**Section 5.1**) and software integration (**Section 5.2**) are explained, followed by a presentation and discussion of the results. The purpose of developing the FPGA-based CFAR detector is achieving low processing latency even for large window sizes while maintaining high energy efficiency, which will be evaluated in **Sections 5.3** and **5.4**.

5.1 Hardware Integration

On AMD Alveo accelerator cards such as the u55c, custom logic processors are integrated as so-called kernels. Depending on available FPGA resources, multiple kernels can be placed on the same chip. Kernels can be connected to each other and to the FPGA-side HBM. As shown in **Figure 5**, the standard PCIe protocol is used to connect the accelerator to the host Central Processing Unit (CPU). Data is exchanged by copying data buffers from the host-side DDR-DRAM to the HBM on the card and vice versa. The host runs a Linux Kernel driver to manage the card and provides an Application Programming Interface (API) for user software.

The CFAR processor was packaged as a kernel using the Vivado 2023.1 software. The kernel was then imported into Vitis 2023.1, where connections between the kernel and the HBM channels are defined. Based on the provided configuration the tool automatically infers glue logic for these connections. The kernel was subsequently synthesized, placed and routed for the target FPGA.

Figure 6 shows the resource utilization of the CFAR processor in default configuration with 300 BRAMs for the pixel cache. The dark blue areas mark cells used by the kernel, while green resources represent other inferred logic. Light blue cells belong to the static region for card management, interfaces etc. The kernel spans three so-called Super Logic Regions (SLRs), which logically and physically separate the FPGA fabric. In fact, SLRs are integrated on separate chips and connected by a silicon interposer. The FPGA on the Alveo u55c comprises three SLRs. The purpose of this Stacked Silicon Interconnect (SSI) technology is increasing the number of available resources and thus allowing for larger circuits. However, crossing the boundaries between SLRs invokes significant signal delay, ultimately reducing the maximum clock frequency that a circuit can run at. The highest clock frequency at which the CFAR kernel with 300 BRAMs is able to reliably meet timing is 150 MHz. The processing latencies discussed in **Section 5.3** will show that the kernel throughput is proportional to its clock frequency, so optimization in this regard is desirable. Analysis of the utilization and signal paths showed that the reason for crossing the SLR boundaries is the number of BRAM cells needed to implement the pixel cache. These cells are evenly distributed across the three SLRs in a column-wise arrangement. If the number of BRAMs can be reduced by a factor of three the whole kernel may fit into a single SLR, potentially increasing the maximum clock frequency.

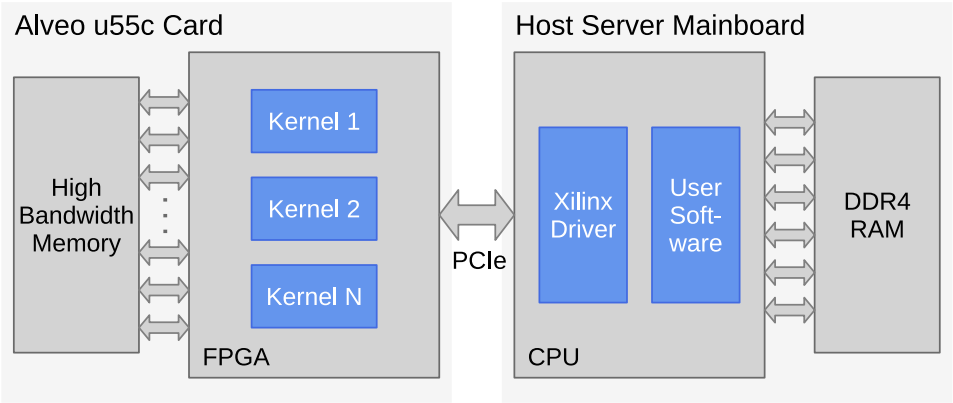

Figure 5 Kernel integration on AMD Alveo u55c. The accelerator card slots into a PCIe socket of the server mainboard. The host CPU runs a Kernel driver for managing the card and provides a programming interface for user software.

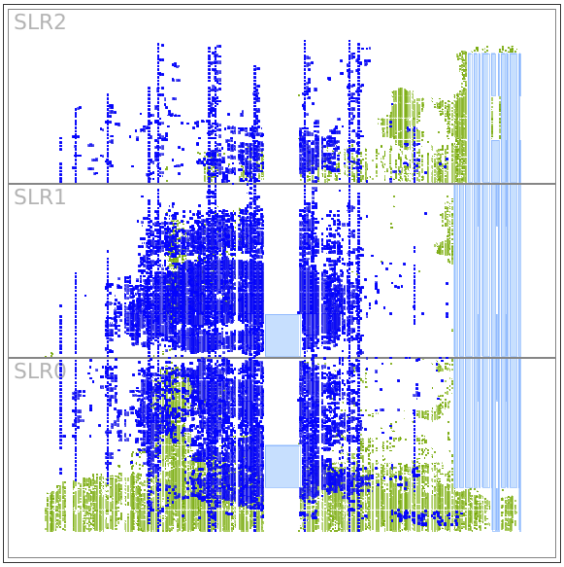

Figure 6 FPGA resource utilization of the CFAR processor with 300 block RAMs for pixel caching.

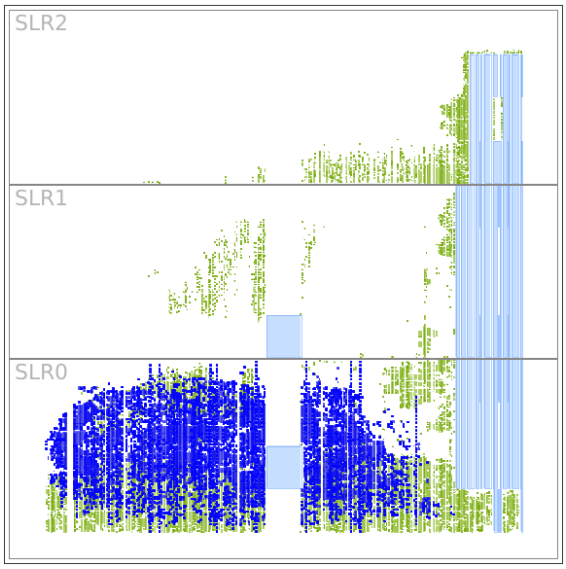

Figure 7 Resource utilization of the dedicated processor for small window sizes using only 90 block RAMs.

The CFAR window pixel cache is dimensioned to support different SAR product types, including high resolution TerraSAR-X StripMap, but for other products, such as the commonly used Sentinel-1 high resolution Interferometric Wide (IW) swath with 10 m pixel size, the CFAR window is usually much smaller. In [3] a typical window size of 750 m or 75 pixels at 10 m pixel size is assumed for reliable ship detection. The number of parallel queues can be configured at compile-time with a single parameter and the multiplexing from the HBM channels to BRAMs changes accordingly. Therefore, with minimal effort a smaller design with just 90 queues was synthesized and the utilization is shown in **Figure 7**. Indeed, the kernel fits into a single SLR and as a consequence the maximum frequency almost doubles to 275 MHz.

5.2 Host Software

To process a SAR image with the CFAR kernel on the Alveo u55c card multiple steps have to be executed using the AMD Xilinx Run-Time (XRT) API. First, the FPGA has to be programmed with the bitstream generated by the Vitis software. The SAR image data has to be partitioned and copied to the HBM according to **Section 4.4**. Several parameters such as image size, CFAR window dimensions, CFAR constant etc. need to be written to metadata registers inside the kernel. Then, the kernel can be started and will signal to the host when it is done. After processing has fin-

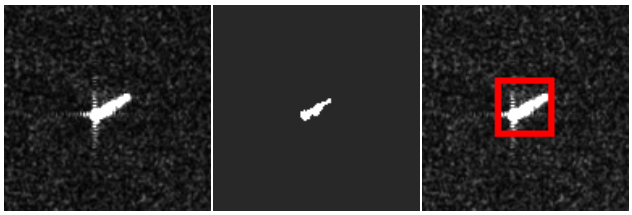

Figure 8 Example of hardware-accelerated CFAR ship detection. Cutout of the pre-processed SAR image (left), binary mask after CFAR kernel (middle) and ship marked by host software after further processing (right).

Table 1 Processing latencies for different CFAR window sizes of the hardware detector compared to Intel Xeon E7-8857 CPU with various amounts of parallel cores.

Win. [px]	FPGA [s]	CPU 12c [s]	CPU 46c [s]
61	7.91	37.2	13.4
75	7.92	44.2	16.2
101	14.53	61.7	25.7
125	14.53	92.9	27.9
151	14.57	122.4	40.4
201	14.60	252.3	65.9
251	14.64	441.7	93.6
299	14.67	871.1	212.7

ished, the binary mask marking detected pixels is copied from HBM back to the host.

A C++ library with a compact API has been developed to automatically perform the outlined steps and shield the user from the low-level driver functions. Either the full-size or frequency-optimized CFAR kernel are automatically picked depending on the window size. With this, the hardware-accelerated CFAR can be easily integrated into existing ship detection software as shown in **Figure 8**.

5.3 Latency

To evaluate the detector with regard to processing latency, the execution time of the kernel was measured. **Table 1** lists, for various CFAR background window sizes, the latencies of the FPGA-based processor compared to a conventional Intel Xeon E7-8857 CPU using different amounts of parallel cores. The processed image is a Sentinel-1 IW high-resolution product with a width of 25927 pixels and height of 16709 pixels (VV polarization). Despite utilization of up to 46 cores, which involves almost 4 full CPUs with 12 cores each, the CPU is slower than the dedicated hardware processor. When the window size is 90 or lower, e.g. in case of Sentinel-1 high resolution products, the frequency-optimized smaller kernel finishes processing in under $8\,\mathrm{s}$. For larger windows (TerraSAR-X etc.), the latency is below $15\,\mathrm{s}$ in all cases.

Looking at the larger kernel, the almost constant latency of the hardware detector regardless of the window size proves that reading the image data from the HBM no longer presents a bottleneck like the DDR-DRAM did in the previous design [3]. The pipeline for calculating the CFAR threshold is fully utilized now, so that the latency is inversely proportional to the clock frequency of the circuit.

5.4 Power Consumption

Executing the CFAR algorithm on the Intel Xeon E7-8857 generates full load on the used cores. The 12-core CPU is specified with a Thermal Design Power (TDP) of $130\,\mathrm{W}$. If the algorithm is processed on four CPUs at the same time as discussed, around $500\,\mathrm{W}$ are required. In comparison, the AMD Alveo u55c reports consumption of only $23\,\mathrm{W}$ to $27\,\mathrm{W}$ total board power during processing, which includes the HBM and other components on the card. The power consumption slightly increases with the CFAR window size because larger windows require more data to be read from the HBM.

The FPGA-based detector is not only faster than a conventional CPU, especially when processing high-resolution products, but also more efficient by an order of magnitude. This makes it potentially suitable for satellite on-board processing as well, where power consumption and heat dissipation are major concerns.

6 Conclusion

In this work, an FPGA-based CFAR processor for ship detection on satellite SAR imagery has been developed. A previous performance bottleneck was eliminated by utilizing state-of-the-art HBM and implementing a sophisticated memory partitioning and multiplexing architecture. The detector has been successfully integrated on an AMD Alveo u55c data center accelerator card and validated in an existing ground-based processing chain, which was facilitated by developing a C++ library for automating the low-level card control and data transfers. The hardware detector is run-time configurable regarding CFAR parameters and can be adapted to different SAR missions and product types. At typical window sizes, a Sentinel-1 IW high-resolution product can be processed in less than 8 seconds, while even higher resolution products such as TerraSAR-X StripMap can be processed in under 15 seconds. Compared to a conventional server-grade CPU, power efficiency has been improved by an order of magnitude. The whole card consumed just $23\,\mathrm{W}$ to $27\,\mathrm{W}$ depending on the configured window size.

7 Literature

[1] R. Cumplido, C. Torres, and S. Lopez, "On the implementation of an efficient FPGA-based CFAR processor for target detection," in *1st International Conference on Electrical and Electronics Engineering (ICEEE).*, pp. 214–218, 2004.

[2] S. Brusch, S. Lehner, T. Fritz, M. Soccorsi, A. Soloviev, and B. van Schie, "Ship Surveillance With TerraSAR-X," *IEEE T. Geosci. Remote*, vol. 49, no. 3, pp. 1092–1103, 2011.

[3] S. Wiehle, S. Mandapati, D. Günzel, H. Breit, and U. Balss, "Synthetic Aperture Radar Image Formation and Processing on an MPSoC," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022.

[4] S. Ghosh, P. K. Konugurthi, G. Shankar Rao Singupurapu, S. Patel, T. Tammanagari, M. Rao Desu, L. K. Thakar, and I. Ghara, "On-board ship detection for medium resolution optical sensors," *Sensors*, vol. 21, no. 9, 2021.

[5] A. Frost, R. Ressel, and S. Lehner, "Automated Iceberg Detection Using High-Resolution X-Band SAR Images," *Can. J. Remote Sens.*, vol. 42, no. 4, pp. 354–366, 2016.