An Energy-Efficient Correlator Architecture design for FPGA

R.A.JAUREGUI MORRIS[1]
[1]Faculty of EEMCS, University of Twente, Enschede 7500 AE, Netherlands
Corresponding author: R.A.Jauregui Morris (e-mail: r.a.jaureguimorris@student.utwente.nl).

ABSTRACT In radio astronomy, large radio telescopes use multiple fields of antennas around the Earth to capture waveforms in the sky for astronomical measurement and imaging. The significance of the correlator in the telescopes is to extract useful information from multiple pairs of antennas in one field and between fields for a higher resolution of the imaging. Therefore, it usually tends to be computationally expensive due to its quadratic growth relative to the number of pairs of antennas in a two-dimensional grid, the number of bands, and frequency channels. The SKA correlator with more than one hundred thousand antennas is predicted to consume in the Mega-Watts range. Assuming power demand grows in proportion to the computational intensity, the power demand of the correlator will grow quadratically as well. Different techniques were presented for optimizing the correlator performance in speed and area. Approximate computing has been one of the techniques to reduce the power budget of high computational expenses. In recent studies, a set of approximate multipliers was employed in the core of the correlators to investigate the quality performance and the reduction of the power demand. A 19% power reduction was achieved by doing so with Application Specific Integrated Circuit (ASIC). However, these types of approximate multipliers are only optimal on ASIC technology and will lack performance on a Field Programmable Gate Array (FPGA) device, due to their architectural differences. This paper presents a methodology for building a correlator architecture from an algebraic description. A set of power Pareto-Optimal 8-bit input approximate multipliers designed for ASIC and FPGA implementation are investigated for their performances concerning the correlator's output-quality for image processing, the number of LUTs savings, and power savings improvement on an FPGA device. For SNR levels at the input (SNR_{in}) of the correlators up to 30 dB, the highest core dynamic power savings achieved by using the approximate multipliers in this paper is 26.56% (see Table 8). However, as the number of antennas increases quadratically for correlation, this power savings decay to 2.9%. For this approximate multiplier, an average of 4.2% number of LUT savings is achieved on the FPGA.

INDEX TERMS Correlator, Approximate Computing, Approximate Multiplier, FPGA

I. INTRODUCTION

In Radio Telescopes, the correlator is a component used to extract useful information from multiple pairs of antennas. This extracted data is further processed by the subsequent component to construct a visual image with a similar angular resolution as an optical telescope [6]. As the pairs of antennas increase, the correlator computational requirement increases quadratically. Large radio telescopes cover large fields of antennas and baselines to increase sensitivity and spatial resolution. Such large radio telescopes are the Square Kilometre Array (SKA) and Low-Frequency Array (LOFAR) [15] [16] [20]. Taking into account the number of bands and the frequency channels, the processing requirements of the correlator will become approximately two orders of magnitude greater than the rest of the processing stages in these types of radio telescopes [21]. Assuming the power consumption grows in proportion to the processing demands, the correlator would always be the highest power-demanding component for large radio telescopes. To quantify the amount, it is predicted that the SKA correlator would have consumed 7.9MW if the same technology as the ALMA correlator had been applied [13].

The core of cross-correlating an antenna pair is based upon a complex multiply-accumulate (CMAC) operation [13]. When the LOFAR was introduced, this operation was performed on many-core processors such as the General Purpose Processors (GPPs) and the Graphics Processing Units (GPUs) [17]–[20]. However, the size and power profiles of these types of processors are dependent on the prediction of Moore's Law and Dennard Scaling [14]. Hence, no energy-efficiency improvement would have been gained on the correlator for large radio telescopes when newer GPPs or GPUs are employed. Therefore, the CMAC operation has been brought to the circuit-level, where Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs) are the prominent devices for the realization. At this level, several techniques were proposed to improve the speed of the correlator on an FPGA for large radio telescopes. For instance, the Single-Instruction-Multiple-Data (SIMD) technique [22]–[24] and the optimized algorithm of rearranging the CMAC operators from the triangular form matrix structure [25]. The SIMD technique takes advantage of the internal pre- and post-adders in the FPGA's hardware multiplier, namely the Digital Signal Processing (DSP), to perform the multiply-add-operation within one execution cycle. Thus, compromising the speed of the CMAC to the latency of the DSP block. The optimized algorithm achieves a speed-up of 4% for the expense of using 14% more of the DSP blocks within a single execution cycle.

Recently, approximate computing has become a prospective technique for reducing the power demands of the correlators on ASIC technology [1] [2]. Approximate computing is a technique in digital circuit designs that exchanges the exactness of computations for an improvement in power dissipation, circuit area, and latency reduction [26]. In general, they are useful in error-tolerant applications such as machine learning, pattern recognition, and image processing. For the radio astronomy correlators, the approximate computing benefits of reducing power dissipation are constrained to the input Signal-to-Noise ratio (SNR_{in}) on the antenna array. The simplified interferometry with a set of SNR_{in} levels on a single point source at the antenna array, different approximate multipliers display different levels of noise correlation on three output-quality metrics of the correlator [1]. These metrics are the Signal-to-Noise-Ratio (SNR_{dB}), the Spurious-Free-Dynamic-Range (SFDR_{dB}) and the Root-Mean-Square (RMS_{dB}). The approximate multiplier with a normalized absolute mean error of 1.57e-8 from [2] has shown a 19% power dissipation reduction for all output-quality metrics up to SNR_{in} of 10dB, and with a higher normalized absolute mean error, a 12% power reduction for SNR_{in} of 30dB. Several past works have illustrated that the approximate computing principles and techniques developed for ASIC implementation tend to offer dissimilar benefits when realized on FPGA-based [29]–[32]. No information is presented in the literature on applying approximate computing techniques for FPGA devices on radio astronomy correlators to reduce power dissipation.

Therefore, the research question of this paper is formulated: "What is the most optimal (complex) multiplier for energy-efficiency improvement on FPGA?" In this paper, the prior work in investigating the feasibility of approximate computing on the radio astronomy correlator model in [1] is extended by realizing the model on an FPGA device. The output-quality mapping and the hardware gains of the implementation with different sets of approximate multipliers for ASIC and FPGA implementation are investigated. An overview of radio astronomy and interferometry, large radio telescopes followed by a survey of on approximate multipliers is presented in Section II. Section III presents the preliminaries on defining area, latency, and power consumption from an FPGA device. The methodology and the circuit-level implementation of realizing the radio astronomy correlator model are presented in Section IV and Section V, respectively. The hardware, power, and output-quality evaluation are presented in Section VI.

II. BACKGROUND

In this section, an overview of radio astronomy telescope processing and correlator architecture is presented with a literature review on techniques to optimize the correlator architecture for energy-efficiency, latency, and hardware resource utilization.

A. RADIO ASTRONOMY AND INTERFEROMETRY

In radio astronomy, the electromagnetic radiations of the celestial object are essential for a telescope instrument to explore the universe. The radio waves with a long wavelength (λ), that can penetrate the earth's atmosphere with little distortion, are of interest to radio astronomers. These radio wavelengths (frequencies) range from 10 meters (30 MHz) to 1 millimeter (300 GHz) [6]. In any optical instrument, the angular resolution is used as the measuring factor to distinguish two separate points with an angular distance for producing the details of the image. The smaller the angular resolution, the finer the detail of the image can be constructed. The angular resolution is derived by the wavelength of observations divided by the size of the instrument. Since radio telescopes operate with longer wavelengths compared to optical telescopes, the physical size of the radio telescope must be much larger to gain an equivalent angular resolution. The interferometry technique was introduced in the radio telescope to alleviate the antenna's physical size for higher angular resolution. Essentially, the technique uses a 2-dimensional grid of antennas, namely an antenna array, at the receiver-end to synthesize an effective aperture of the captured wavelength. The captured signal from one antenna will be in- or out of phase with the separate antenna in the interferometry plane, due to the differing path lengths from the source. The antennas in the interferometry plane produce interference fringes of the source in a similar way conducted on an optical interferometer.

These interference fringes in radio telescopes are produced by multiplying and time-averaging the captured wavelength voltages from a pair of antennas. The circuity that performs the latter is essentially the correlator [7]. Mathematically, the response of the cross-correlation from one pair is described as [9],

$$C(\tau) = \frac{1}{T}\int_0^T V_i(t)V_j^*(t-\tau)\,dt \qquad (1)$$

where V_i and V_j are the waveforms at the correlator input from antennas i and j, τ is the time by which the waveform V_j is delayed with respect to the waveform V_i, the superscript asterisk indicates the complex conjugate and T the integration length. For signal analysis, the captured waveforms on the antennas are assumed to be independent and identically distributed (i.i.d.) random variables, due to their long distance from the celestial source. This causes the cross-correlation to act as an auto-correlation operation. Maintaining τ to zero or close to zero produces the maximum interference fringes from the captured signal between a pair of antenna.

For an array of antennas, the two methods to control τ are the phased array and the correlator array [10]. A phased array is realized using analog instruments such as a voltage combiner followed by a square-law detector to output the total power of all the antennas. It also uses an adjustable phased shifter on every antenna to control τ. By doing so the beam-pattern of the antenna array is controllable to scan an area of the sky. A more complex signal-combining network is required to form many more beams. The scanned areas in the sky are then further processed to form an image. On the other hand, the correlator array responds to the whole field of the individual antennas. Here, the measured cross-correlations are combined e.g. a supercomputer or a many-core processor, with appropriate phase variations to form different beam patterns for scanning the sky. Therefore, the beam-forming of the correlator array collects more data efficiently for image processing compared to the phased array.

In image processing, the output of the correlator from the set of antennas is considered to be in the *visibility* domain, which has a uv-plane with a u-axis pointing to the east of the earth and a v-axis to the north [8]. Measuring the cross-correlation of a pair of antennas is assumed to be a two-dimensional visibility component denoted as $\mathcal{V}(u, v)$ where the distance between them is represented as a baseline [9]. Taking the two-dimensional Inverse Fourier Transformation on the visibility components within the uv-plane produces the two-dimensional intensity distribution, denoted as $\mathcal{I}(l, m)$, which is the synthesis image of a field of the sky. The higher the number of $\mathcal{V}(u, v)$ components is given in the uv-plane, the higher the resolution of the synthesis image can be constructed. On the contrary, the higher the power dissipation becomes assuming that power grows in proportion to the number of cross-correlation operations.

B. LARGE RADIO TELESCOPES

To quantify the magnitude of the power dissipation, the correlator for large radio telescopes is constructed in triangular form matrix structure for an arbitrary set of antenna arrays across different continents around the Earth. For instance, the SKA is split between two large radio telescopes, namely the Australian Square Kilometre Array Pathfinder (ASKAP) and the Karo Array telescope (MeerKAT), which are also known as SKA1-Low and SKA1-Mid, respectively. The SKA1-Low consists of 131072 dipole antennas, distributed in 512 stations, where each station is grouped into 256 antennas with a baseline of 65 km and set to target signals at a frequency range between 50 MHz to 350 MHz. The SKA1-Mid is set to target a frequency range between 350 MHz to 15.3 GHz and contains 190 of 15m dishes, and 64 of 13.5m dishes with baselines up to 150 km [16]. The processing stage, which conducts the cross-correlation operation, within the processing pipeline of the SKA1-Low has the highest processing requirement with 1.51 peta operations per second [15] [21]. Assuming power grows in proportion to the number of operations, the SKA1-Low correlator stage will always have a greater power consumption than the rest of the processing stages. Applying approximate computing on the SKA correlator with the same processing requirement may gain a lower power dissipation for less accurate computation. To investigate this, the modeling of the SKA is ignored and the simplified interferometry in [1] is used since it is difficult to model the SKA processing requirement and conduct power measurement properly without a large number of antennas. The simplified interferometry consists of a square plane of antennas (A$\times$A) with one reference antenna at the center and A set to be an even number. All the antennas in the antenna array are spaced $\frac{1}{2}\ \lambda$ from each other. The reference antenna is set as the origin relative to the rest of the antennas. The cross-correlation of a single-pair antenna is a modular operation on the signal received from the reference antenna with every antenna in the antenna array A$\times$A.

C. SURVEY ON APPROXIMATE MULTIPLIER

In this section, a survey of the existence of approximate multipliers methodology for the simplified interferometry in [1] is conducted.

The approximate multipliers are designed in such a way as to construct approximate MAC (xMAC) accelerators, where the multiplier's output errors are constrained to a near-to-zero mean profile so that the accumulator can cancel out the produced errors. Gillani *et. al* [2] proposed the internal-self-healing (ISH) methodology on the xMAC accelerators, where a recursive multiplier (RM) structure is optimized to an approximate RM (xRM) that can produce a near-to-zero mean error profile for a given input distribution. The internal streams of errors in xMAC are an additive inverse of each other so that the overall mean error approaches zero. Appendix A elaborates on the construction of the xRM using the RM structure as a base.

Even though an improvement is achievable regarding power reduction and reduction in area, these xRMs may not gain the same performance improvement when deployed on FPGA, due to the architectural differences between FPGA and ASIC. Any approximate circuit design for an FPGA device must consider the FPGA internal core structure to gain the approximate computing technique benefits, e.i. the reduction of power demand, latency or on number of hardware resource usage [27]–[32]. For this reason, studies have been conducted on designing approximate multipliers for FPGA-based. Appendix B gives a summary of the core structure of Xilinx's FPGA and Intel's FPGA.

In SMApproxLib [27] and [28], the proposed approximate multipliers are essentially a reduced version of hardware usage from their distinct 4-bit multipliers base architecture. Both base architectures exploit the core structure of Xilinx FPGA by using the LUT6_2 and carry chain primitives. The higher-order bit-width of both FPGA-based approximate multipliers requires the RM structure approach, where the sub-multipliers are a set of 4-bit multipliers. In [27], three 8-bit xRM are proposed as a bit-width extension of the proposed 4-bit approximate multipliers. In [28], a set of 8-bit xRM is proposed which is a combination of its proposed 4-bit approximate multipliers. The proposed 8-bit xRM from both papers are not eligible to be deployed on the xMAC accelerator, since their sequence of errors is not additive inverse of each other, and no knowledge of the type of the input distribution is presented.

The Pareto-optimal sets of xRM in [29] and [30] consider the ISH-methodology constrained, where one set is optimal for lowering the hardware utilization and the other for lowering the power demand on an FPGA device.

Rick van Loo [29] proposed the Approxy framework to produce the Pareto-optimal sets of 4-bit xRM for FPGA devices. The framework uses the same set of elementary 2-bit approximate multipliers from [2] for the architectural-space exploration. Using the Xilinx Kintex-7 FPGA and a Normal Distributed (N(μ,σ^2)) input with a mean (μ) of 8 and standard deviation (σ) of 1.5, the optimal 4-bit xRM achieve 25% improvement on power compare to the 4-bit accurate RM.

Rienk van der Wijk [30] proposed a Pareto-optimal set of 8-bit xRM with four 4-bit multipliers. The design-space for deploying the sub-multipliers on the recursive multiplier structure consists of the Pareto-optimal sets from the Approxy framework [29], two 4-bit approximate multipliers and one 4-bit accurate multiplier from the SMApproxLib [27]. Using Xilinx Kintex-7 FPGA and a Normal Distributed input of N(128, 22.5^2), the optimal 8-bit xRM design is estimated to produce $516\mu W$ with a Normalized Absolute Mean Error of 7.19e-4. However, the estimated power result is not verified by the synthesis tool. Regarding hardware utilization, the optimal 8-bit xRM consists of 37 LUTs with a Normalized Absolute Mean Error of 3.78e-2. Comparing this result with the optimal 8-bit approximate multiplier from SMApproxLib [27], 9 LUTs were saved.

Another way to construct the xMAC accelerators as a whole is by using the Xel-FPGA framework [31] where a Pareto-Optimal hardware utilization set of approximate accelerators is extracted from a design-space of 1000 random ASIC-based approximate circuit variants. The framework exploration method is pipeline within three stages: the statistical regression model training, the architectural exploration, and the final evaluation. During the model training, two statistical regression models are being trained. One for estimating the output-quality and one for estimating power, hardware utilization, and latency, of every approximate accelerator on Xilinx Virtex-7 FPGA. After the statistical models are trained, the architectural exploration stage uses the NSGA-II genetic [42] algorithm to select and iterative explore the architectural-space of the accelerators. The selected accelerators are further synthesized and simulated in the final evaluation stage to determine their accurate hardware requirement and output-quality. Using the NSGA-II genetic algorithm and the logic synthesis ABC-tool [41], which is used to extract the hardware accelerators features in the model training stage, allows the framework to achieve an exploration time of roughly 5 hours with a Pearson Correlation Coefficient of 0.9 for a design-space of 1 million approximate accelerator variants.

Xiang *et al.* [32] proposes an approximate logic synthesis (ALS) method for transforming the local LUT sub-networks into approximate ones with the minimum numbers of LUTs on an FPGA with a given LUT input size. The method uses the Fanout-Free Cone (FFC) structure in the LUT network, which implements a single-output function through the LUT sub-network. The approximate design thus consists of minimizing the number of LUTs needed to satisfy the FFC structure on the LUT subnetworks. The reduction uses disjoint decomposition on a given Boolean function with a single-output. For functions that are not decomposable, the approximate disjoint decomposition is used to find a decomposable function close to it. The latter thus transforms the accurate Boolean function to a Boolean function with some errors. An iterative process of the algorithm is required for design with multiple-output, since the technique is restricted only to single-output function. The proposed approximate logic synthesis methodology has been experimented on benchmark circuits suite from EPFL[1] [43] and MCNC[2] using the logic synthesis ABC-tool [41]. The proposed approximate logic synthesis method only focuses on constructing an approximate circuit design with error rate as the error metric and misses the average error magnitude. Since the average error magnitude of the 2-bit input multipliers is required in a xRM to build a near-to-zero mean error profile, the ALS technique requires further investigation to construct approximate multipliers to be deployed on xMAC accelerators.

D. MOTIVATIONAL ANALYSIS

Currently, the output-quality of the correlator array structure in [1] is collected for the 8-bit xRM ASIC implementation in [2]. No studies have addressed the output-quality metrics of the correlator array embedded with 8-bit xRM FPGA implementation. Furthermore, most of the research on constructing the Pareto-optimal xRM FPGA implementation is synthesized on modern Xilinx FPGA core structure, which primarily uses adaptive LUTs (LUT6_2) with carry-chain primitives. No studies have addressed Pareto-optimal xRM designs on Intel's FPGA Core Structure. It is expected, that the xRM FPGA-based designs, will yield a different hardware performance, regarding latency and the number of LUTs used, for a different LUT-based architecture.

[1]https://github.com/lsils/benchmarks
[2]https://github.com/lsils/SCE-benchmarks/tree/main/MCNC

TABLE 1: Synthesis-based comparison of a few Pareto-optimal configurations for 8-bit input recursive multipliers, that employ the conventional and the Internal-Self-Healing (ISH) methodologies using the elementary 2-bit multipliers in Appendix A as building blocks. The conventional approximate computing methodology was restricted to a design space of {M, M1, M2} and the ISH to {M, M1, M2, M3, M4}. This table is adapted from [2].

Design	**Pareto-optimal multiplier configurations optimized for normally distributed input**																
Alternatives	**LSM***							$\rightarrow$								**MSM***	**Error***
Accurate	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	0
Conven1 [2]	M1	M1	M1	M	M	M1	M	M1	M	M	M1	M1	M	M	M	M1	2.95e-5
Conven2 [2]	M	M	M	M	M	M1	M	M	M	M	M	M	M	M	M	M1	1.87e-6
ISH1 [2]	M4	M1	M1	M1	M1	M1	M4	M1	M1	M1	M1	M1	M3	M4	M1	M4	1.57e-8
ISH2 [2]	M4	M1	M1	M	M4	M4	M3	M1	M1	M	M4	M1	M	M1	M3	M1	9.26e-9
FPGA_ISH1 [30]	M1	M1	M3	M1	M1	M1	M3	M1	M1	M1	M3	M1	M1	M1	M3	M1	7.19e-4
FPGA_ISH2 [30]	M1	M1	M3	M1	M1	M1	M3	M1	M1	M1	M3	M1	M	M1	M3	M1	1.18e-4
FPGA_ISH3 [30]	M1	M1	M3	M1	M1	M1	M3	M1	M	M1	M3	M1	M	M3	M3	M4	3.47e-5
FPGA_ISH4 [30]	M1	M1	M3	M1	M	M1	M3	M1	M	M1	M3	M1	M	M3	M3	M4	2.47e-6
FPGA_ISH5 [30]	M	M1	M3	M1	M	M1	M3	M1	M	M1	M3	M1	M	M3	M3	M4	5.74e-7

* The resulting Error is based on normally distributed input of N(128,22.5^2). The normalized absolute mean error is considered as an Error. LSM and MSM are the least significant and the most significant 2-bit input multipliers respectively.

In this thesis, for investigating the most energy-efficiency correlator architecture design on FPGA, the correlator model of the simplified interferometry in [1] is realized on an FPGA with the Pareto-optimal power optimized sets of 8-bit input xRM. The Pareto-optimal power optimized sets from [2] and [30] use the same set of 2-bit elementary multipliers from Appendix A while having a different configuration for constructing xMAC. Since this difference configuration is due to the architectural difference between ASIC and FPGA, both sets of xRMs will be synthesized on Intel's FPGA for hardware performance comparison, regarding the number of LUTs usage, latency, and power consumption. Table 1 illustrates the Pareto-optimal power-optimized sets for investigation. Since the conventional approximate computing methodology was restricted to a design-space of {M, M1 M2}, their performance on hardware, latency, and power on the CMACs may have higher gains compared to the ISH methodology. To quantify the hardware performance gains in proportion to the number of CMACs used, the correlator array with sizes of 1×1, 2×2, and 4×4 is presented. Appendix F investigates more in-depth the latency performance of the xRMs, when embedded in the correlator architecture and realized on an FPGA.

III. PRELIMINARIES

A. POWER MEASUREMENT

Intel Quartus Prime Standard provides the possibility to run a gate-level timing simulation of a post-synthesis circuit design, where the switching activity and glitches for all cells are tracked during the run. However, this timing simulation is only supported for Arria II GX/GZ, Cyclone IV, MAX II, MAX V, and Stratix IV Intel device families [40]. After running the gate-level timing simulation, a *Value-Change Dump* (VCD) file is generated, which contains the switching activity on all nodes during the run. Providing this VCD file to Intel Quartus Power Analyzer, the total power, core dynamic power dissipation (P$_{\text{dynamic}}$), and the core static power dissipation (P$_{\text{static}}$) is estimated. Appendix C elaborates the details on P$_{\text{dynamic}}$ and P$_{\text{static}}$ on FPGA.

IV. METHODOLOGY

To realize the correlator array of the simplified interferometry in [1], the derivation of the Signal Flow Graph (SFG) from its algebraic description is used. Appendix D elaborates more in detail on deriving the SFG from an algebraic description.

A. ANALYSIS OF THE ALGORITHM

Assuming the antenna signals are digitized and complex (Re and Im), the core of the cross-correlation can be defined as a complex multiply-accumulate (CMAC) operator [13],

$$c_{i,j} = \sum_{t=0}^{T-1} v_{i,t} v_{j,t}^{*} \tag{2}$$

where v_i and v_j are the discrete waveforms from antenna i and j, respectively. In the visibility domain, the algebraic equivalence of $c_{i,j}$ is described as $\mathcal{V}(v_i, v_j)$. For the simplified interferometry, the input v_i is substituted to $a_{i,j}$ and v_j to r, hence

$$c_{i,j} = \sum_{t=0}^{T-1} a_{i,j,t} \cdot r_t^{*} \tag{3}$$

where $a_{i,j}$ is the discrete waveform from antenna coordinated at (i, j) in the antenna array A×A and r from the reference antenna. The distribution of the inputs to each CMAC operator follows Algo. 1. The correlator array outcome for such a case produces a two-dimensional visibility component shown in Eq. 4 and A^2 baselines.

Algorithm 1 Pseudo-code of the simplified interferometry.

```
1: for $i = 1 : A$ do
2:     for $j = 1 : A$ do
3:         $\mathcal{V}(a_{i,j}, r)$
4:     end for
5: end for
```

$$\mathcal{V} = \begin{pmatrix} c_{1,1} & c_{2,1} & c_{3,1} & \cdots & c_{A,1} \\ c_{1,2} & c_{2,2} & c_{3,2} & \cdots & c_{A,2} \\ c_{1,3} & c_{2,3} & c_{3,3} & \cdots & c_{A,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{1,A} & c_{2,A} & c_{3,A} & \cdots & c_{A,A} \end{pmatrix} \tag{4}$$

B. ALGORITHM TO SIGNAL FLOW GRAPH

The dependency graph (DG) from Eq. 3 is a 3-dimensional structure due to the existence of 3-indexes. To minimize the DG to a 2-dimensional structure, we map the subscript i, j to one index. This mapping follows the escalation of the subscript i, j as shown in the Algo. 1. For the correlator array of N×N, we will substitute A in the Algo. 1 to N. Taking the escalation of subscript i, j and the substitution of A to N, the subscript i, j is mapped as,

$$\begin{aligned} (1,1) &\mapsto x_1 \\ (1,2) &\mapsto x_2 \\ &\vdots \\ (1,N-1) &\mapsto x_{N-1} \\ (1,N) &\mapsto x_N \\ (2,1) &\mapsto x_{N+1} \\ (2,2) &\mapsto x_{N+2} \\ &\vdots \\ (N,N-1) &\mapsto x_{N\times N-1} \\ (N,N) &\mapsto x_{N\times N} \end{aligned}$$

where x_n denotes the nth element of the sequence x. To build the local DG from Eq. 3, r_t^* is substitute to $d_{x_{n-1},t}$, where $d_{x_{n-1},t}$ is equal to $d_{x_n,t}$ and $d_{x_0,t}$ to r_t^*. We will raise the bounds in the summation by one for cosmetics on drawings, hence Eq. 3 is rewritten to,

$$c_{x_n} = \sum_{t=1}^{T} a_{x_n,t} \cdot d_{x_{n-1},t} \tag{5}$$

From this reformed equation, the recurrent relation is derived as,

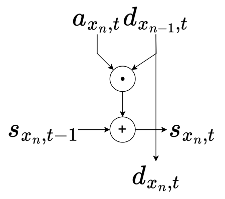
$$s_{x_n,t} = s_{x_n,t-1} + a_{x_n,t} \cdot d_{x_{n-1},t} \tag{6}$$

The maximum bound of the recurrent relation is $s_{x_n,T}$, which is equal to c_{x_n}, and the minimum bound is $s_{x_0,0}$, which is set to zero. Fig. 1 illustrates the derivation of the two-dimensional DG based on the recurrent relation in Eq. 6 with the maximum and minimum bounds. From the internal structure of the DG vertex, there are two paths between the inputs and outputs. The path with the longest computational delay, e.i. the critical path, in the DG vertex consists of one multiplication and one addition. From the DG and the internal structure of the DG vertex, we can observe the critical path consists of one multiplication and T number of additions.

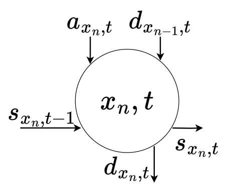
To fold DG to the SFG, we select the direction of the projection vector **d** equal to the direction of increasing t subscript, e.i. **d** = (0,1). The other direction of **d** folds the DG to an SFG, where the integration length becomes a factor in forming SFG vertices. Hence, the design architecture, which is implemented on the FPGA, has to be redesigned for every change of the integration length. Moreover, we consider the synchronous machine behavior of the SFG vertices to be mealy machines. Thus, the outputs of the SFG vertex are defined as,

$$y_t, s_t = F_{\text{mealy}}(x_t, s_{t\text{-}1}) \tag{7}$$

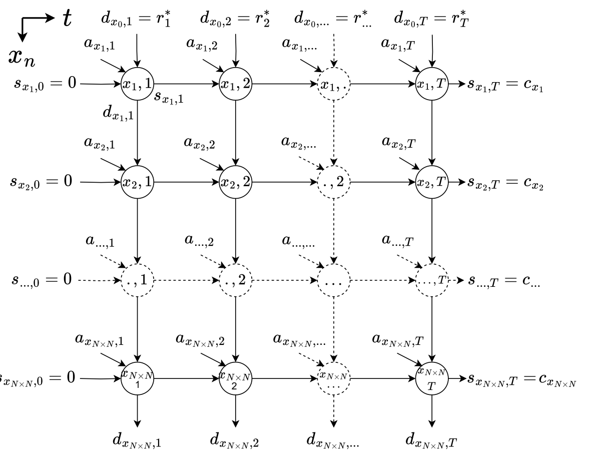
where s_t records the output state of the SFG vertex at time instance t, and the entry $s_{t\text{-}1}$ takes the output state at t-1. From the recurrent relation in Eq. 6, the F_{mealy} is a multiply-add operation. Thus, the integration length becomes a factor in placing multipliers and adders to compensate the number of SFG vertices, that are mapped from the DG. With the projection vector **d** set equal to (0,1) and the processor assignment **P** being a null-space vector to **d**, **P** is set to (1,0). The two-dimensional DG in Fig. 1 is folded to the SFG by applying **w** = **P** · **v**, where **w** is the set of vectors, e.i. vertices, intermediate edges, input edges, and output edges, for the SFG and **v** for the DG.



(a) The internal structure of the DG vertex.



(b) The DG vertex representation.



(c) The DG representation with the minimum bound $s_{x_n,0}$ set equal to zero and the maximum bound $s_{x_n,T}$ set equal to c_{x_n}.

FIGURE 1: The dependency graph (DG) from the recurrent relation in Eq. 6 for a given antenna array of N×N and integration length of T. The internal structure of the DG vertex follows Eq. 6 with $d_{x_n,t}$ set equal to $d_{x_{n\text{-}1},t}$.

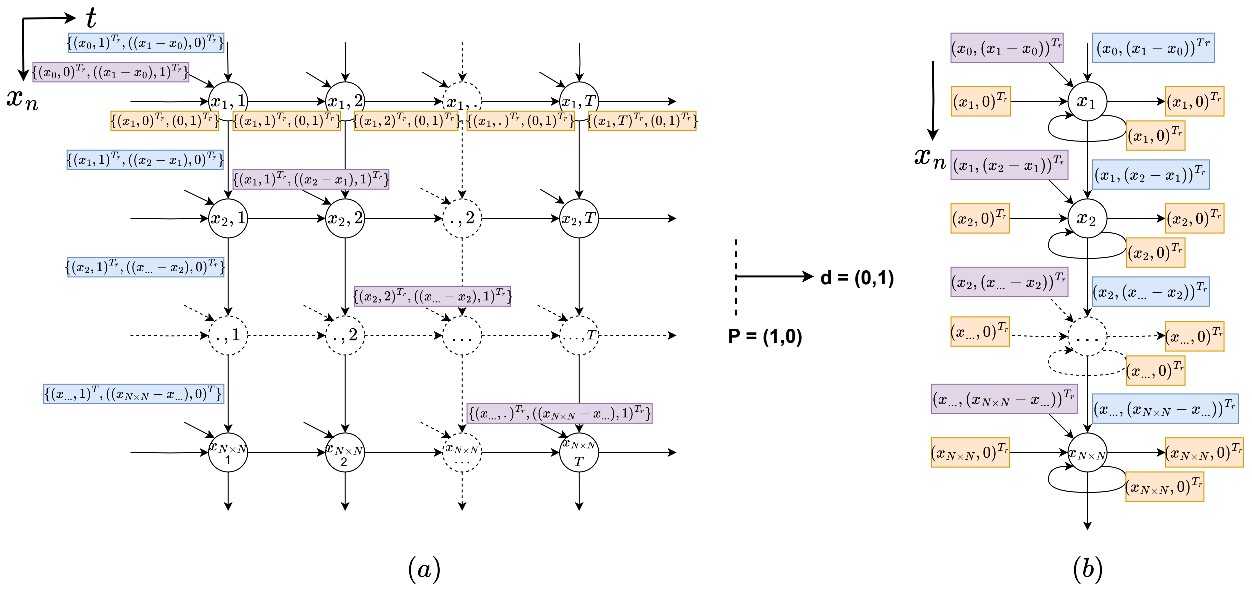


FIGURE 2: The mapping of the dependency graph (a) to signal flow graph (b) by using the processor assignment **P** set to (1,0).

For instance, a set of vertices in **v** described as $\{x_n, t\}$ is mapped to the SFG described as,

$$\begin{aligned}\mathbf{w}_{\text{vertex}} &= \mathbf{P} \cdot \mathbf{v}_{\text{vertex}} && (8)\\ &= (1,0) \cdot \{x_n, t\} && (9)\\ &= \{x_n\} && (10)\end{aligned}$$

The set of intermediate edges in **v** from a vertex $\{x_n, t\}$ with a displacement vector $\mathbf{e}_\text{d}$ of $\{(x_n\text{-}x_{n\text{-}1},0)^{T_r}\}$, e.i. the vertical edges of the DG, are mapped to the set of intermediate edges in **w** as,

$$\begin{aligned}\{\mathbf{w}_{\text{vertex}}, \mathbf{e}_d\} &= \mathbf{P} \cdot \{\mathbf{v}_{\text{vertex}}, \mathbf{e}_d\} && (11)\\ &= (1,0) \cdot \{(x_n, t)^{T_r}, (x_n - x_{n\text{-}1}, 0)^{T_r}\} && (12)\\ &= \{(x_n, x_n - x_{n\text{-}1})^{T_r}\} && (13)\end{aligned}$$

Note T_r indicates the transpose transformation of a matrix. The set of intermediate edges in **v** from a vertex $\{x_n, t\}$ with $\mathbf{e}_\text{d}$ of $\{(0,1)^{T_r}\}$, e.i. the horizontal edges of the DG, are mapped to the set of looping edges in **w** from the vertex $\{x_n\}$. The set of input edges in **v** from a vertex $\{x_0, t\}$ with $\mathbf{e}_\text{d}$ of $\{(x_1\text{-}x_0,0)^{T_r}\}$, e.i. the vertical input edges of the DG, are mapped to the input edge in **w** from vertex $\{x_0\}$ with $\mathbf{e}_\text{d}$ of $\{(x_1\text{-}x_0)^{T_r}\}$, e.i. the vertical input edge of the SFG. The set of input edges in **v** from a vertex $\{x_n, 0\}$ with $\mathbf{e}_\text{d}$ of $\{(0,1)^{T_r}\}$, e.i. the horizontal input edges of the DG, are mapped to input edges in **w** from vertex $\{x_n\}$ with $\mathbf{e}_\text{d}$ of $\{(0)^{T_r}\}$, e.i. the horizontal input edges of the SFG. The set of input edges in **v** from a vertex $\{x_{n\text{-}1}, t\text{-}1\}$ with $\mathbf{e}_\text{d}$ of $\{(x_n\text{-}x_{n\text{-}1},1)^{T_r}\}$, e.i. the diagonal input edges of the DG, are mapped to input edges in **w** from vertex $\{x_{n\text{-}1}\}$ with $\mathbf{e}_\text{d}$ of $\{(x_n\text{-}x_{n\text{-}1})^{T_r}\}$, e.i. the diagonal input edges of the SFG. Fig. 2 illustrates the mapping of the vectors in the DG to the SFG by using the processor assignment **P** set equal to (1,0). Additionally, the mathematical descriptions of the edges in the DG and the SFG are shown.

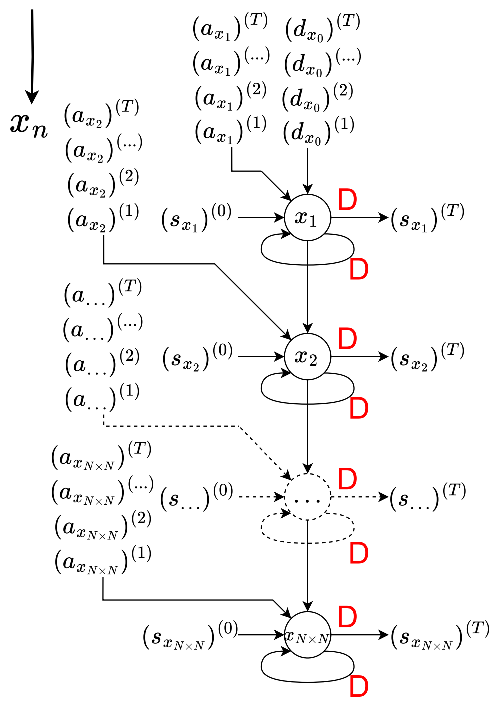
The scheduling of the data produced by the DG vertices in time instance t to the SFG vertices follows the product of the scheduling vector **s** with the source vertex $\mathbf{v}_\text{source}$ (Appendix D). However, the scheduling vector $\mathbf{s}^{T_r}$ is restricted from the $\mathbf{e}_\text{d}$ in the DG edges and the selected projection vector **d**. The scheduling vector is restricted clockwise from (0,1), e.i. an increase on the subscript t, to (1,1), e.i. an increase on subscript sequence x_n and t. Since the number of unit delays (D^n) on the SFG edges is dependent on the displacement vector in the DG with the scheduling vector as, n = **s** · $\mathbf{e}_\text{d}$ (Appendix D), the scheduling vector **s** set to (0,1) allocates D at all the SFG edges with a $\mathbf{e}_\text{d}$ equal to zero, except for the input edges. For instance, the intermediate and output edges of the DG with the displacement vector $\mathbf{e}_\text{d}$ of $\{(0,1)^{T_r}\}$ computes D^n on their corresponding mapped SFG edges as,

$$\begin{aligned}n &= \mathbf{s} \cdot (\mathbf{v}_{\text{dest.}} - \mathbf{v}_{\text{source}}) && (14)\\ &= \mathbf{s} \cdot \mathbf{e}_d && (15)\\ &= (0,1)(0,1)^{T_r} = 1 && (16)\end{aligned}$$

The scheduling vector **s** set to (0,1) allocates the least number of unit delays in the SFG edges, which conserves power dissipation. With the selected scheduling vector **s** of (0,1), the data produced and consumed by the DG vertices can be scheduled to the SFG. For example, the data $d_{x_0,1}$, $s_{x_1,0}$ and $a_{x_1,1}$ in Fig. 1c is scheduled to be consumed by the SFG vertex $\{x_1\}$ at time instance,

$$t = \mathbf{s} \cdot \mathbf{v}_{\text{source}} = (0,1) \cdot (x_0, 1) = 1 \quad (17)$$

Similarly, $d_{x_1,1}$, $s_{x_2,0}$ and $a_{x_2,1}$ is scheduled to be consumed



x_n
$(a_{x_1})^{(T)}$ $(d_{x_0})^{(T)}$
$(a_{x_1})^{(\dots)}$ $(d_{x_0})^{(\dots)}$
$(a_{x_1})^{(2)}$ $(d_{x_0})^{(2)}$
$(a_{x_1})^{(1)}$ $(d_{x_0})^{(1)}$
$(a_{x_2})^{(T)}$
$(a_{x_2})^{(\dots)}$
$(a_{x_2})^{(2)}$
$(a_{x_2})^{(1)}$
$(s_{x_1})^{(0)}$
x_1
$(s_{x_1})^{(T)}$
D
$(a_{\dots})^{(T)}$
$(a_{\dots})^{(\dots)}$
$(a_{\dots})^{(2)}$
$(a_{\dots})^{(1)}$
$(s_{x_2})^{(0)}$
x_2
$(s_{x_2})^{(T)}$
$(a_{x_{N\times N}})^{(T)}$
$(a_{x_{N\times N}})^{(\dots)}$
$(a_{x_{N\times N}})^{(2)}$
$(a_{x_{N\times N}})^{(1)}$
$(s_{\dots})^{(0)}$
$(s_{\dots})^{(T)}$
$(s_{x_{N\times N}})^{(0)}$
$x_{N\times N}$
$(s_{x_{N\times N}})^{(T)}$

FIGURE 3: The fully annotated signal flow graph (SFG) from Fig. 2 with the time-coordinate information of the data and delay unit (D). The annotation $(y)^{(t)}$ on the edges indicates the time instance (t) at the data y takes place. These annotations follow the data dependency in Fig. 1c for the scheduling vector **s** set to (0,1).

by the SFG vertex $\{x_2\}$ at time instance,

$$t = \mathbf{s} \cdot \mathbf{v}_{\text{source}} = (0, 1) \cdot (x_1, 1) = 1 \tag{18}$$

These two examples show that the SFG vertices consume and produce data in parallel. Fig. 3 illustrates the annotated scheduling of all the data that the SFG vertices consume with the number of delay units on the output edge and the looping edge. The output edge at vertex $\{x_{N\times N}\}$ is eliminated since the SFG vertex $\{x_{N\times N}\}$ is the last vertex that requires the data of the reference antenna represented as $d_{x_0,t}$. This fully annotated SFG demonstrates the graphical representation of distributing the stream of the reference antenna, which is represented as $\{(d^*_{x_0})^1, \dots, (d^*_{x_0})^T\}$, to the SFG vertices $\{x_n\}$ for processing with the stream of the antennas $\{(a_{x_n})^1, \dots, (a_{x_n})^T\}$ in parallel.

Recall that the SFG vertices have a mealy machine behavior with F_{mealy} being a multiply-add operation. The delay unit on both outputs of the SFG vertices consumes the output result of F_{mealy}. The delay unit on the looping edges stores the result s_t at a time instance t and returns it to their corresponding SFG vertices entry $S_{t\text{-}1}$. With the add operation in the vertices, these delay units accumulate the result of the SFG vertices as long as the integration length (T). Therefore, combining an SFG vertex and the delay unit (D) on the looping edge has a multiply-accumulator (MAC) behavior. Since the antenna signals are complex, the actual behavior is a CMAC behavior.

The multiplication of two complex inputs can be realized using the Cartesian method, which expresses a polynomial expansion [23] as,

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc) \tag{19}$$

To illustrate the requirement of realizing the CMAC behavior of the SFG vertex in combination with the delay unit on their corresponding looping edge, the two multiplicands in Eq. 5 is instantiated as complex, and the Cartesian method is employed for the complex multiplication. Hence, Eq. 5 is expanded to,

$$c_{x_n} = O_{1,x_n} - O_{2,x_n} + i(O_{3,x_n} + O_{4,x_n}) \tag{20}$$

where

$$O_{1,x_n} = \sum_{t=1}^{T} \text{Re}\{a_{x_n,t}\} \otimes \text{Re}\{d_{x_{n\text{-}1}}\} \tag{21}$$

$$O_{2,x_n} = \sum_{t=1}^{T} \text{Im}\{a_{x_n,t}\} \otimes \text{Im}\{d_{x_{n\text{-}1}}\} \tag{22}$$

$$O_{3,x_n} = \sum_{t=1}^{T} \text{Re}\{a_{x_n,t}\} \otimes \text{Im}\{d_{x_{n\text{-}1}}\} \tag{23}$$

$$O_{4,x_n} = \sum_{t=1}^{T} \text{Im}\{a_{x_n,t}\} \otimes \text{Re}\{d_{x_{n\text{-}1}}\} \tag{24}$$

The $\otimes$ notation refers to the approximate multipliers employed.

C. THE CORRELATOR ARRAY STRUCTURE

Digitization of the signal waveform requires sampling of the voltages at periodic intervals and quantizing the sampled values so that each can be represented by a finite number of bits [11]. This process is conducted by the Automatic-Gain-Control (AGC) and the Analog-to-Digital converter (ADC) components. The function of the AGC is to adapt the signal's voltage amplitude at the RMS level so that the maximum amplitude of the ADC is defined. This optimally exploits the dynamic range of the ADC, where the RMS level is used for de-normalization purposes [1]. In contrast, the ADC does the sampling of the signal at periodic intervals. The de-normalization of the two correlated inputs in Eq. 20 is computed as,

$$z_{x_n} = c_{x_n} \cdot \sigma_{a_{x_n}} \cdot \sigma_r \tag{25}$$

where $\sigma_{a_{x_n}}$ and σ_r are the complex standard deviations of the discrete waveform of a_{x_n} and r, respectively, which are measured from the AGCs. The produced two-dimensional visibility component of the correlator array (Eq. 4) entries are substituted to z_{x_n}.

Fig. 4 illustrates the simplified interferometry of 2×2 antenna array connected to the realization of the SFG from

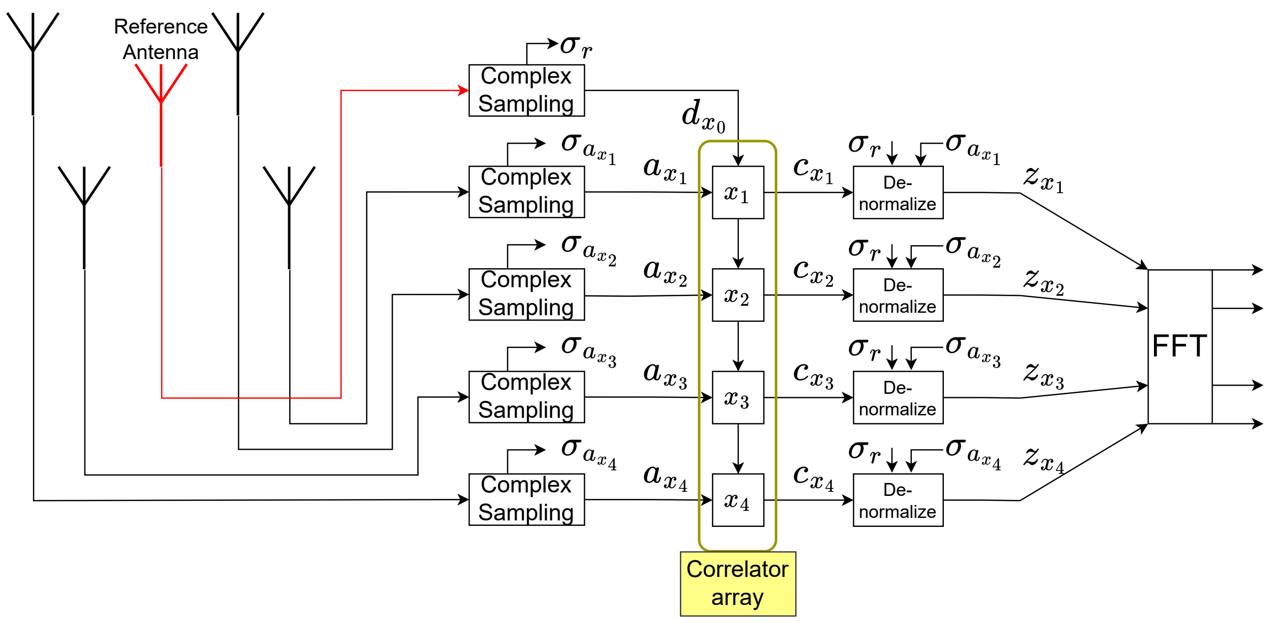


FIGURE 4: The simplified interferometry of 2×2 with one reference antenna connected to a correlator array of 2×2. The structural information of the correlator array is similar to the SFG structure from Fig. 3. The complex sampling blocks digitize the signals from the antennas with their corresponding complex standard deviations. The de-normalize block takes the complex standard deviation of their corresponding correlated signals.

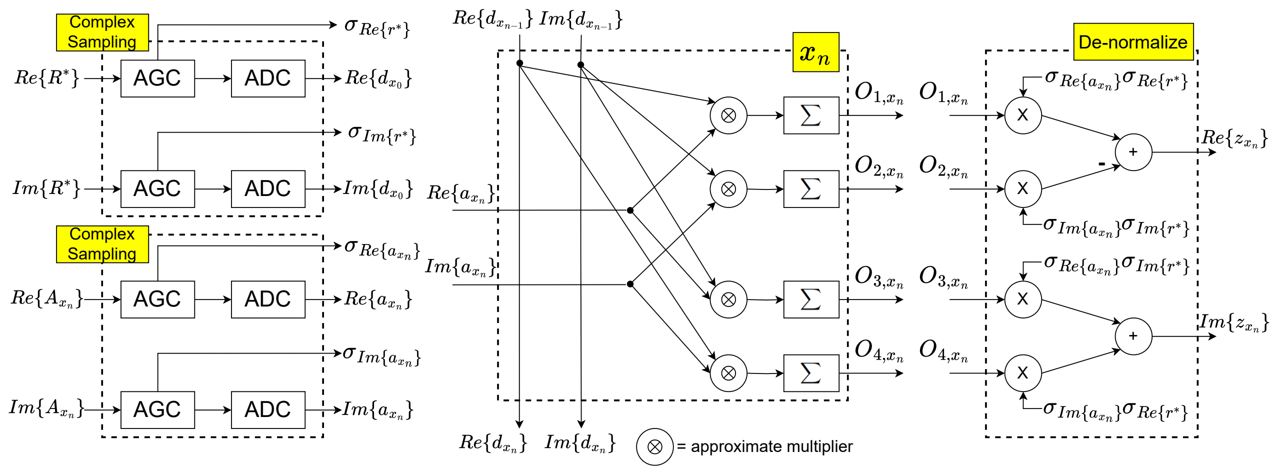


FIGURE 5: The internal structure of the complex sampling, complex correlation x_n, and the de-normalization block.

Eq. 5, e.i. expanded as complex correlation, the complex sampling of the antennas signals, and the de-normalization of c_{x_n}. The internal structure realization of each component is illustrated in Fig. 5. The outputs of the de-normalized correlation z_{x_n} are further processed by the two-dimensional Fast Fourier Transform to produce the intensity distribution $\mathcal{I}(l, m)$. In this paper, the correlator array is implemented on the FPGA, whereas the complex sampling and the de-normalized blocks are conducted on higher-level programming software. The digitized sample, that is produced by the ADC, is a 9-bit sign-magnitude data representation. This way each multiplier calculates the sign of a product based on the sign bits of its inputs in parallel the magnitudes of the inputs are multiplied by an 8-bit unsigned multiplier structure, which can be instantiated to an approximate multiplier. The circuit-level realization of the CMAC behavior x_n block in combination with the latter is further elaborated in Section V.

D. SIPO AND PISO

An FPGA device is limited to realizing correlator array structure with a large number of antennas due to their physical

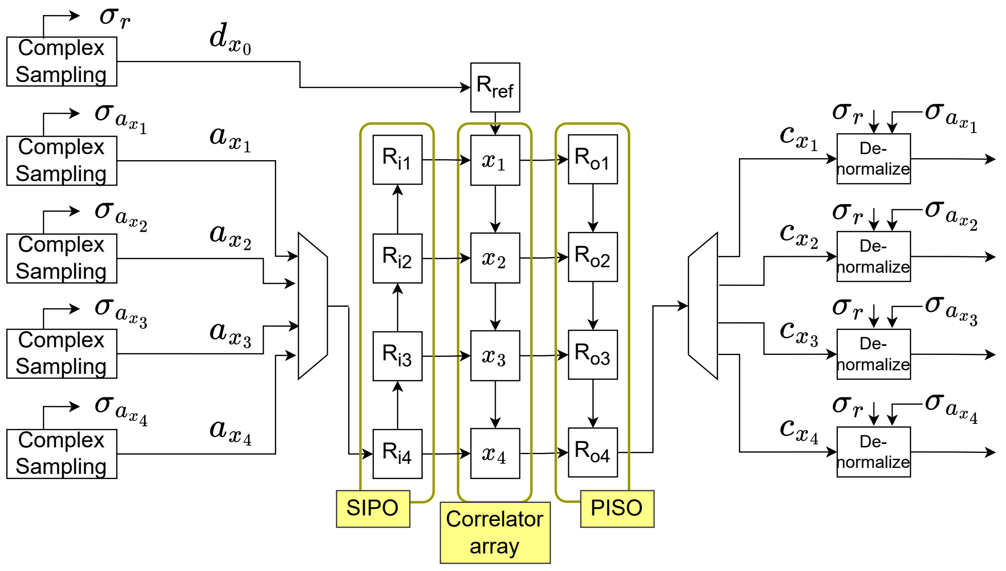


FIGURE 6: Added the SIPO and PISO shift registers on the inputs and outputs of the correlator array for the same antenna array in Fig. 4.

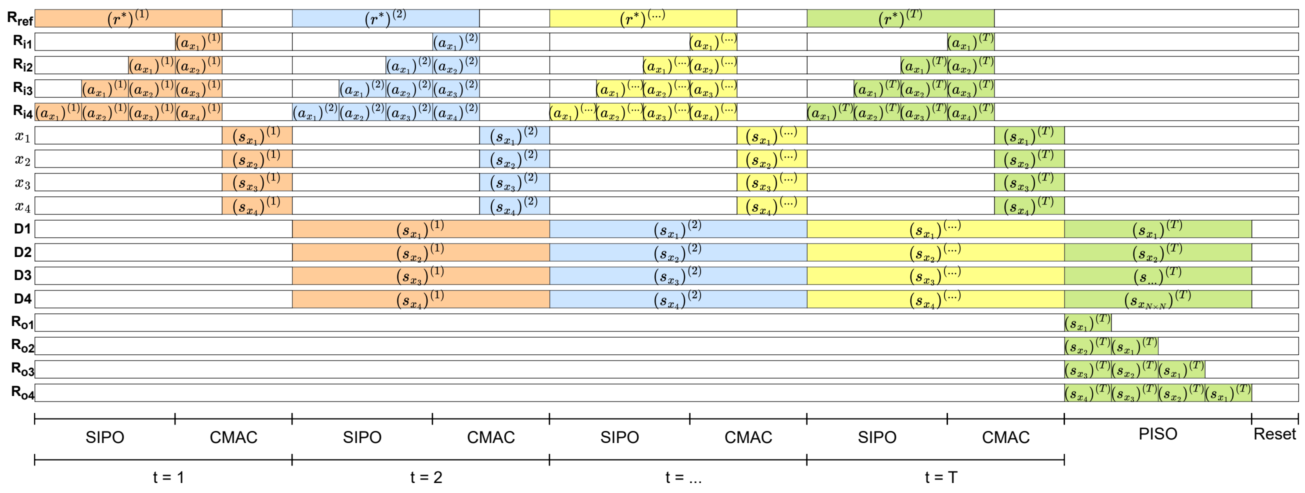


FIGURE 7: The timing diagram of SIPO and PISO shift registers with the correlator array from Fig. 6.

limitation on the available pins. The proposed optimization for this is to employ shift registers at the expense of an increase in latency and power consumption. The Serial-in Parallel-out (SIPO) shift register is employed on the inputs of the correlator array minimizing the number of input pins from an arbitrary set of antennas a_{x_n} to the number of input pins of only one a_{x_n}. To mitigate the number of output pins from an arbitrary set of x_n into the number of output pins of one x_n, the parallel-in Serial-Out (PISO) shift register is employed. Fig. 6 illustrate the SIPO and PISO shift registers placement between the correlator array in Fig. 4.

The Serial-in Parallel-out (SIPO) shift register shifts the set of streams $\{(a_{x_1})^t, \ldots, a_{x_{N\times N}})^t\}$ to its corresponding input edges so that the SFG vertices consumes and produces in parallel at an arbitrary time instance t. It is established from the SFG in Fig. 3, that the SFG vertices, e.i. $\{x_1, \ldots, x_{N\times N}\}$, indeed consumes data in parallel on the input edges at a time instance t. Since the set of streams of the antennas input $\{(a_{x_1})^t, \ldots, a_{x_{N\times N}})^t\}$ is scheduled to a particular time instance t, the number of vertices added in the correlator array, increases the latency of the SIPO. This also holds for the PISO shift registers. From the SFG in Fig. 3, the set of streams $(s_{x_1})^T, \ldots, a_{x_{N\times N}})^T\}$ is essentially the multiply accumulate operation at the integration length T. The PISO operation is to shift out these set of streams to the next processing stage. Fig. 7 illustrates the timing diagram of processing the correlator array with an integration length T and SIPO and PISO shift registers for the simplified interferometry setup in Fig. 4.

To evaluate the correlator array power performance on a large antenna array of A×A, the SIPO and PISO shift registers may be used as presented in Fig. 6. For such a case, the correlator array of N×N is limited to processing a subset of antennas in A×A, thus requiring a repetitive operation for different subsets in A×A. This limit follows the ratio between A×A and N×N. For each subset of antennas in A×A feed in the SIPO shift registers in combination with the correlator array and the PISO shift registers, must follow the time-coordinate of the SFG in 3 to compute the complex multiply-accumulation correctly. In other words, the first samples of all the antennas in the subset are fed at time instance one, the second sample at time instance two, the third sample at time instance three, and so on until the integration length T (see Fig. 7). After the sample of the antennas at T is consumed by the CMAC operators, the reset of the accumulations occurs, so that the next subset of antennas in A×A is fed in the correlator array of N×N. This procedure continues until all the antennas in A×A are fed. This way the latency of producing the two-dimensional A×A visibility component is computed as

$$L_{A\times A} = \frac{A^2}{N^2} \cdot T \cdot (t_{\text{SIPO}} + t_{\text{CMAC}}) \qquad (26)$$

where t_{SIPO} and t_{PISO} are the time duration of SIPO and PISO shift registers, respectively.

V. IMPLEMENTATION

For the circuit-level realization, all edges are represented as sign-magnitude data, since this data representation is produced by the ADC. The sign-magnitude data representation uses the Most-Significant-Bit (MSB) as the sign-bit to indicate whether the data is a positive value (sign-bit is set to 0) or a negative value (sign-bit is set to 1). The remainder of the bits represent the data magnitude.

From the multiplication behavior, it is known that two equal sign numbers result in a positive number and two impartial sign numbers a negative. Thus, the resultant sign-bit of the multipliers is in the function of their inputs' sign-bit. Table 2 illustrates this functionality in a truth-table, where $F_1(z_1, z_2)$ represent the resultant sign-bit of the multiplier, and z_1 and z_2 the multiplier input's sign-bit. Note the behavior is similar to an XOR gate. The magnitude bits of

TABLE 2: The sign-bit behavior of multiplying two sign-magnitude data representations.

z_1	z_2	$F_1(z_1, z_2)$
0	0	0
0	1	1
1	0	1
1	1	0

the inputs are consumed by an unsigned multiplier structure. The resultant output of this structure becomes the output magnitude bits. Since the data width of the ADC is 9-bit, the inputs' data widths of the multipliers are equal. The sign-bits of the inputs are connected to a two-bit LUT (2-LUT), which has the same content shown in Table 2. The 8-bit magnitude bits of the inputs are consumed by an 8-bit input unsigned multiplier structure. Hence, the output data width of the multipliers are 17-bit sign-magnitude. The 8-bit unsigned multiplier structure is instantiated by the xRM structures in Table 1 for investigation. Fig. 8 illustrates the hardware circuit realization for the multipliers.

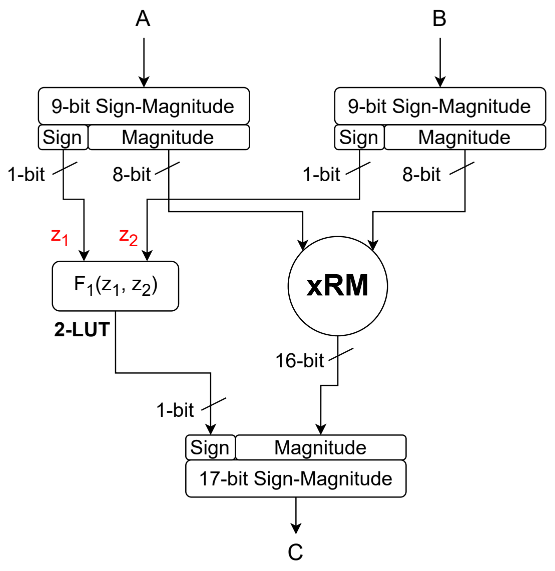


FIGURE 8: The circuit-level realization of the multiplication operation denoted as $\otimes$ from the CMAC behavior of x_n in Fig. 5.

For the circuit-level realization of the adder and accumulator, the magnitude data width of the multiplication operation is extended with zeros to match the output data width (W-bit) of the accumulator. This way both input operands of the addition operation are equivalent. The accumulator data width must correlate with the integration length (T). Combining the input data width from the ADC with the integration length (T) of the accumulation, the W-bit of the accumulator is defined as,

$$W = (\text{INPUT\_BIT\_WIDTH} \times 2 - 1) + \lceil \log_2(T) \rceil \qquad (27)$$

Fig. 9 illustrates the circuit-level realization of the accumulation process, where a register is used as the accumulator and the addition/subtraction operator is assigned as *Addi*.

From the addition/subtraction behavior, originally two positive numbers will result in a larger positive number and two negative numbers a larger negative number. Given the signs of both numbers are different, the resultant sign will be equal to the sign that has the highest value. Additionally, the highest value is also subtracted from the lowest between both numbers. Table 3 illustrates the addition/subtraction functionality in a truth-table for two arbitrary sign-magnitude data representations (A and B). The ripple-carry-adder (RCA) circuit with a subtraction capability is realized

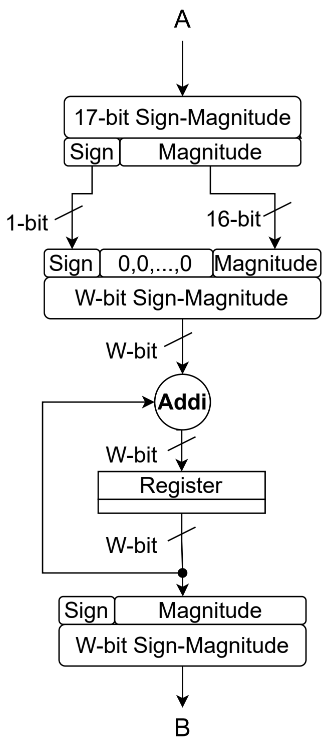


FIGURE 9: The circuit-level realization of the accumulation process denoted as Σ from the CMAC behavior of x_n in Fig. 5.

TABLE 3: The addition/subtraction behavior for two sign-magnitude data representation.

A's sign-bit (z_1)	B's sign-bit (z_2)	A's magnitude $\geq$ B's magnitude (z_3)	addition/ subtraction behavior	output's sign-bit $F_2(z_1, z_2, z_3)$
0	0	0	A+B	0
0	0	1	A+B	0
0	1	0	B−A	1
0	1	1	A−B	0
1	0	0	B−A	0
1	0	1	A−B	1
1	1	0	A+B	1
1	1	1	A+B	1

for the addition/subtraction operation since it uses the least hardware resources compared to other adder designs. From Table 3, the output's sign-bit can be derived in a 3-LUT as $F_2(z_1, z_2, z_3)$, where z_1 denotes to A's sign-bit, z_2 to B's sign-bit and z_3 to A's magnitude greater than or equal to B's magnitude. Moreover, using only one ripple-carry-adder (RCA) with a subtraction capability, two extra LUTs can be derived. One for selecting the input's magnitude bits of A, or B, to the RCA's inputs (x and y), and one for selecting the type of operations (addition or subtraction). Note that for the latter, the selection is dominant only by A's and B's sign-bits and has the same behavior as F_1 from Table 2, where an addition operation occurs for equal sign-bits and a subtraction for impartial. Table 4 illustrates the two truth-table derivations, where one input of RCA is dependent on the outcome of $F_3(z_1, z_2, z_3)$ and the other to its inverse. Fig.

TABLE 4: The truth-table for selecting the input's magnitude bits of A and B to the inputs of the ripple-carry-adder (RCA). z_1 and z_2 represents the sign-bits of input A and B, respectively. z_3 is the outcome of the A's magnitude bits greater than B's magnitude bits.

(z_1)	(z_2)	(z_3)	RCA's input x $F_3(z_1, z_2, z_3)$	RCA's input y $F_3(z_1, z_2, z_3)$	RCA's add/sub $F_1(z_1, z_2)$
0	0	0	A	B	0
0	0	1	A	B	0
0	1	0	B	A	1
0	1	1	A	B	1
1	0	0	B	A	1
1	0	1	A	B	1
1	1	0	A	B	0
1	1	1	A	B	0

10 illustrates the circuit-level realization of *Addi* for two sign-magnitude data representations.

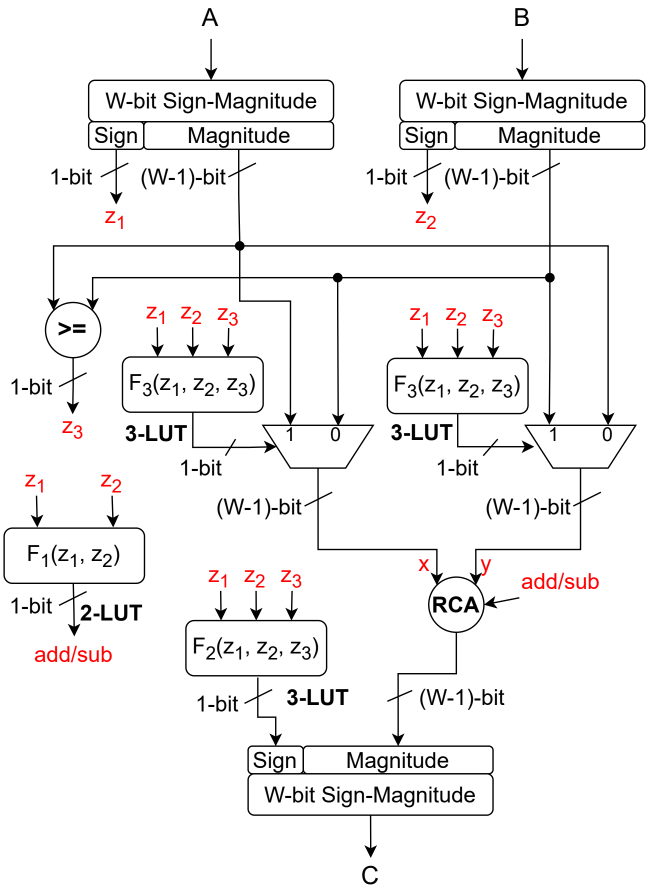


FIGURE 10: The circuit-level realization of *Addi* in Fig. 9. The behavior of F_1, F_2, and F_3 are shown in Table 2, 3 and 4. The addition/subtraction operation is conducted by the ripple-carry-adder (RCA).

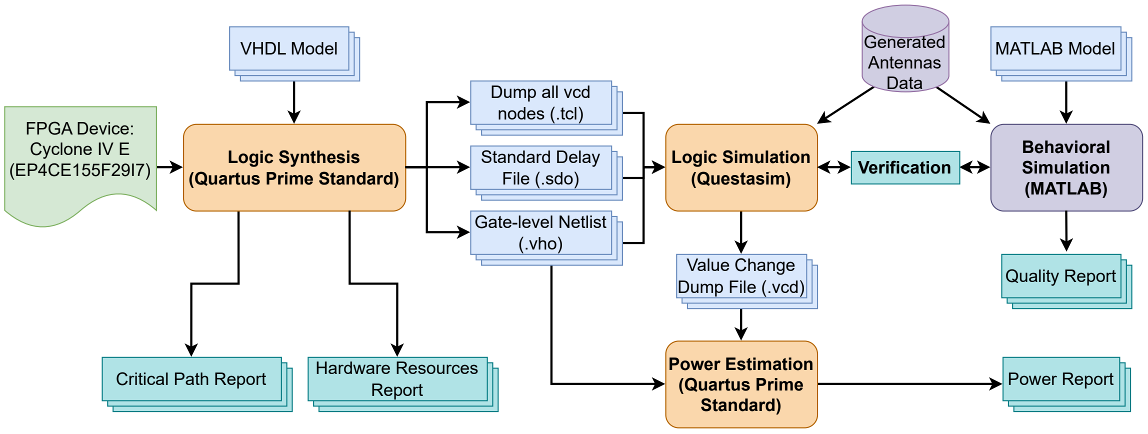

VHDL Model
FPGA Device: Cyclone IV E (EP4CE155F29I7)
Logic Synthesis (Quartus Prime Standard)
Dump all vcd nodes (.tcl)
Standard Delay File (.sdo)
Gate-level Netlist (.vho)
Generated Antennas Data
MATLAB Model
Logic Simulation (Questasim)
Verification
Behavioral Simulation (MATLAB)
Value Change Dump File (.vcd)
Quality Report
Critical Path Report
Hardware Resources Report
Power Estimation (Quartus Prime Standard)
Power Report

FIGURE 11: The overall test setup for collecting hardware resource and power result adapted from [2].

VI. RESULTS

A. EXPERIMENTAL SETUP

The circuit realization of Fig. 8, Fig. 9, and Fig. 10 are described in VHDL. The connection of these components follows the internal structure of x_n in Fig. 5. The synthesis tool for the experiment is the Intel Quartus Prime Standard and the FPGA is the Intel Cyclone IV E *EP4CE155F29I7* as illustrated in Fig. 11. This FPGA device consists of 114480 Logic Elements (LEs), where each of these LEs contains four-input and three-input LUT types. The correlator array in Fig. 4 is synthesized and configured, e.i. Place and Route, on the FPGA with a register placed on the inputs. These registers in combination with the accumulators are used to evaluate the critical path delay and latency of the design on the FPGA (see Appendix F). The signals relevant to the registers, namely the clock, reset, and enable signals, are assigned to the FPGA's physical pins. The clock signal is constrained to 25 nanoseconds (40 MHz). This operating frequency has allowed the FPGA to maintain positive slack and avoid setup time violation for the correlator array of 1×1, 2×2, and 4×4 with an integration length of 64. Fig. 11 illustrates the overall experimental setup for reporting hardware resource utilization, power, and output quality.

B. HARDWARE EVALUATION

The number of LUTs usage, power, and latency results from a xRM are compared against the results of the corresponding accurate RM incorporated in the CMAC operators (x_n) for an integration length of 64. This comparison is also conducted on the correlator array with a size of 1×1, 2×2, 4×4 to quantify the savings. The inputs and output ports of the design are assigned to virtual pins to avoid the constrain of the FPGA's pins during synthesis and configuration. Taking bit-width into account, the total virtual input pins are expected to be the product of the number of SFG vertices, the number of inputs (n_i) of the SFG vertices, and the ADC's output bit-width (9-bit). The total virtual output pins are expected to be the product of the number of SFG vertices, the number of outputs (n_o) of the SFG vertices, and the W-bit (see Eq. 27).

Table 5 presents the number of registers and pins used by the correlator array in Fig. 4 after synthesis. It can be observed that the registers follow the number of the I/O virtual pins, which follows the assignment of the registers to all I/O ports.

TABLE 5: The hardware resource report, regarding registers and pins used on the FPGA for the correlator array architecture 1×1, 2×2, 4×4, 6×6, 8×8 and 10×10 in Fig. 4 and integration length of 64.

N×N	Registers	Pins	Virtual Pins
1x1	128	3	128
2x2	458	3	458
4x4	1778	3	1778
6x6	3978	3	3978
8x8	7058	3	7058
10x10	11018	3	11018

Table 6 presents the number of LUTs used for realizing the correlator array of 1×1, 2×2, 4×4, 6×6, 8×8 and 10×10 with the 8-bit input xRMs in Table 1 and Fig. 12 the number of LUTs savings of using the xRMs relative to the accurate RM. It is expected that as N increases linearly, the size of the correlator array increases quadratically (N×N), and thus the number of LUTs usage also increases quadratically. From Table 6, it can observed that this is the case. With the accurate RM, the correlator array of 1×1 uses roughly 0.89% of the maximum available LUTs in the FPGA device. For the correlator array of 2×2, 4×4, 6×6, 8×8 and 10 ×10 with the accurate RM, roughly 3.57%, 14.26%, 32.13%, 57.14%, and 89.03% is used, respectively, from the maximum available LUTs in the FPGA device. From Fig. 12, it can be observed that the number of LUT savings (%) is not a constant number across different sizes of the correlator array. Therefore, it cannot be claimed that the number of LUTs savings on an FPGA for a giving xRM design is a definite number. However, it can be claimed that it is approximate to a mean value. From the literature, the Conv1, Conv2, ISH1, and ISH2 are the Pareto-optimal 8-bit input xRM designed on ASIC. Taking

TABLE 6: The number of LUTs used after synthesis for the correlator array of 1×1, 2×2, 4×4, 6×6, 8×8 and 10 ×10 with the 8-bit input recursive multiplier structures in Table 1 incorporated.

Array Size	1×1	2×2	4×4	6×6	8×8	10×10
Accurate	1023	4093	16333	36791	65420	101925
Conv1	987	3944	15770	35551	63316	98584
Conv2	1013	4047	16149	36364	64759	100928
ISH1	964	3846	15389	34684	61682	96186
ISH2	978	3896	15584	35074	62561	97456
FPGA_ISH1	959	3832	15294	34421	61416	95757
FPGA_ISH2	960	3841	15394	34631	61687	96174
FPGA_ISH3	969	3877	15538	34956	62153	96947
FPGA_ISH4	975	3899	15588	35085	62429	97361
FPGA_ISH5	979	3912	15635	35222	62708	97785

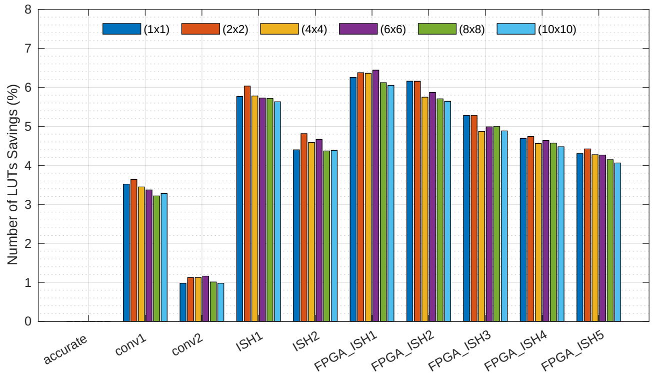


FIGURE 12: The number of LUTs savings of using the *xRM* relative to the accurate *RM* for correlator array sizes in Table 6.

the mean value of the LUTs savings across the correlator array size of 1×1, 2×2, 4×4, 6×6, 8×8, 10×10 (see Table 7), the two conventional approximate multipliers have the lowest percentage of LUTs savings with a mean of 1.06% for Conv2 and 3.42% for Conv1. The two internal self-healing approximate multipliers designed for ASIC have the number of LUTs savings relative to the accurate multiplier with a mean of 5.78% for ISH1 and 4.54% for ISH2. The lowest number of LUTs savings of the approximate multiplier with the ISH methodology employed is the FPGA_ISH5 with a mean of 4.24%, where the highest is the FPGA_ISH1 with a mean of 6.27%.

TABLE 7: The mean value across the correlator array size of 1×1, 2×2, 4×4, 6×6, 8×8, 10×10 for the 8-bit input *xRM* structures in Fig. 12

xRM structure	μ (%)
Conv1	3.42
Conv2	1.06
ISH1	5.78
ISH2	4.54
FPGA_ISH1	6.27
FPGA_ISH2	5.88
FPGA_ISH3	5.05
FPGA_ISH4	4.61
FPGA_ISH5	4.24

Appendix F evaluates the latency of the correlator array architecture with the set of the xRMs in Table 1 and the critical path on the FPGA.

C. POWER EVALUATION

The power estimation is computed by Intel Quartus Power Analyzer where the toggle rates are set to derived from a VCD file. The tcl script, e.i. dump all vcd nodes, the standard delay file, and the gate-level netlist are generated from Quartus Prime Standard after synthesis and configuring the design on the FPGA (see Fig. 11). Similar to the Synopsys Design Compiler output for ASIC, the gate-level netlist is the synthesized design on FPGA written in VHDL, and the standard delay file is associated with this netlist. The *tcl* script is required for recording all signals during gate-level timing simulation. A testbench with a test-vector-controllor (TVC) is additionally written on VHDL to be utilized on Questasim for the functional behavior verification and gate-level timing simulation. After synthesis of the design on the FPGA, the gate-level netlist functional behavior is verified by the resemblance of its produced accumulation vectors with the behavioral model on MATLAB. The antenna input vectors for the functional behavior verification are generated from the data of all antennas with SNR$_{\text{in}}$ set equal to 10 dB, the quantization type of the ADC set to mid-treat, and the AGC set to 5σ-clipping. The captured waveform from an antenna array is generally spread to all antennas, where each antenna produces a noise component and a shifted version of the source component. The antenna array of the simplified interferometry is set equal to the correlator array size. The noise component $n_{i,j,t}$ and the source component s_t are set to a complex Gaussian distribution with zero mean and a variance of one $(\mathrm{CN}(0,1))$ since this assumption is mostly used in practice for modeling the distribution of the captured signals [11] [12]. The length of the antenna input vectors is equivalent to the integration length (T). The simulation is set to terminate by the TVC component after the gate-level netlist of the design processes the last content of the antenna input vectors. Appendix E elaborates more on the testbench setup and the test flowchart. Power dissipation was collected for correlator array of 1×1, 2×2 and 4×4. For all three cases, the antenna array A×A is set equal to the correlator array N×N. The xRM structure for power evaluation on hardware are presented in Table 1,

The static power dissipation of the FPGA is reported at roughly 100 milliwatts (mW), regardless of the circuit-design size. This is expected since the core static power dissipation is dominated by the current leakage present in the internal structure of the FPGA device [33]–[35]. The I/O power dissipation is reported at 30 mW for all correlator array sizes and 8-bit input RM structures. This is due to the similar occurrences of the switching activity on the three physical pins assigned, namely the clock, reset, and enable signals, for all circuit design. These three control signals are provided by the TVC component, which shows the behavior of the test is the same for all correlator array sizes and the 8-bit input RM structures. The switching activity of the signals assigned on the virtual pins is omitted on the I/O power dissipation report since the synthesis tool maps virtual pins to LUTs.

Table 8 illustrates the core dynamic power dissipation ($P_{dynamic}$) report of the correlator array and its savings by using xRM structure relative to the accurate RM and Fig. 13 illustrate the plot of the core dynamic power savings. The correlator array sizes are 1×1, 2×2, and 4×4. From Table 8,

TABLE 8: The dynamic power dissipation ($P_{dynamic}$) report from the power analyzer for the correlator array of 1×1, 2×2, and 4×4 with the *RM* structure in Table 1 incorporated. The power savings are given in percentage (%) and computed by using the *xRM* relative to the accurate *RM*.

Multiplier Type	1×1 (mW)	1×1 (%)	2×2 (mW)	2×2 (%)	4×4 (mW)	4×4 (%)
Accurate	4.6	0	12.41	0	46.68	0
Conv1	4.17	9.35	13.79	-11.12	46.8	-0.26
Conv2	4.95	-7.61	13.04	-5.08	47.55	-1.86
ISH1	3.76	18.26	13.08	-5.4	47.69	-2.16
ISH2	4.19	8.91	13.42	-8.14	46.97	-0.62
FPGA_ISH1	3.83	16.74	12.67	-2.1	45.93	1.61
FPGA_ISH2	4.55	1.09	13.66	-10.07	47.51	-1.78
FPGA_ISH3	5.14	-11.74	13.34	-7.49	46.81	-0.28
FPGA_ISH4	3.97	13.69	12.67	-2.09	47.67	-2.12
FPGA_ISH5	3.38	26.52	12.05	2.9	47.96	-2.74

it can be observed that the core dynamic power dissipation increases quadratically as the correlator array size increases quadratically. The reason for this rise is due to the dynamic power dissipation of the FPGA dependency on the number of LUTs utilization [34], which quadruples in proportion to the correlator array size (see Table 6). It can be observed from

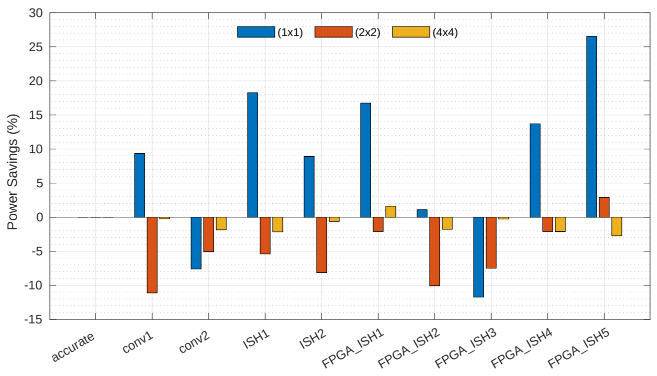


FIGURE 13: The core dynamic power savings in percentage (%) of the correlator array of 1×1, 2×2 and 4×4 embedded with the set of the 8-bit input *RM* in Table 1.

Fig. 13, that the power savings result of the correlator array of 1×1, 2×2 and 4×4 varies for the set of xRMs structure embedded. From Table 8, the correlator array with the size of 1×1 achieves a negative power reduction for Conv2 and FPGA_ISH3 xRM structure, while the rest gain a positive savings result. The highest power reduction achievable on the correlator array of 1×1 is the FPGA_ISH5 with 26.52%. With the size of 2×2, the correlator array achieves a positive power savings for FPGA_ISH5 xRM structure and a negative for the rest. The highest power reduction achievable for the correlator array of 2×2 is the FPGA_ISH5 with 2.9%. With the size of 4×4, the correlator array achieves a positive power reduction for using the FPGA_ISH1 xRM structure and a negative for the rest. The highest power reduction achievable for the correlator array of 4×4 is the FPGA_ISH1 with 1.61%. It can be observed that the correlator array of 1×1 with one of the xRM embedded, namely ISH1, FPGA_ISH1, FPGA_ISH4, and FPGA_ISH5, gain a power reduction that is higher than 10%. As soon as the size of the correlator array increases, the highest power savings become 2.9%. In Appendix F, we measured the time duration increase on the internal critical path of the FPGA as the correlator array size increased. This is due to the congestion on the routing track in the FPGA between LUTs, which the synthesis tool maps a longer physical path between the LUTs on the FPGA to fit the circuit design. The capacitive load on this path may dissipate more energy than the shorter paths and thus more dynamic power is consumed [35]. Therefore, the power savings performance of using the power Pareto-optimal 8-bit input xRM ISH1, FPGA_ISH1, FPGA_ISH4, and FPGA_ISH5, on the correlator array of 1×1 is exploited and decays greatly as the correlator size increases in the FPGA.

The total power dissipation report from Quartus is essentially the sum of the core static, dynamic, and I/O power dissipation. Fig. 14 illustrates the total power savings result of using the 8-bit input xRM relative to the accurate RM. It can be observed, that for all correlator array sizes with all the xRM, the total power savings have a similar form as the core dynamic power savings in Fig. 13. However, the magnitude of the savings is between -1% to 1%. The reason for this is due to static power dissipation is roughly at 100 mW for all cases, which is higher than the dynamic power, and thus the static power of the FPGA lowers the power savings of using xRM to less than 1%.

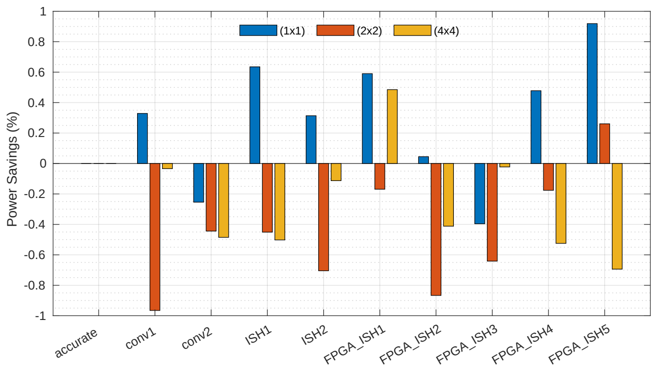


FIGURE 14: The total power dissipation savings in percentage (%) of the correlator array of 1×1, 2×2 and 4×4 embedded with the set of 8-bit input RM in Table 1.

D. OUTPUT-QUALITY

For the output-quality metrics, the correlator behavior in producing the synthesis image is evaluated. This synthesis image is derived by taking the two-dimensional Inverse Fourier

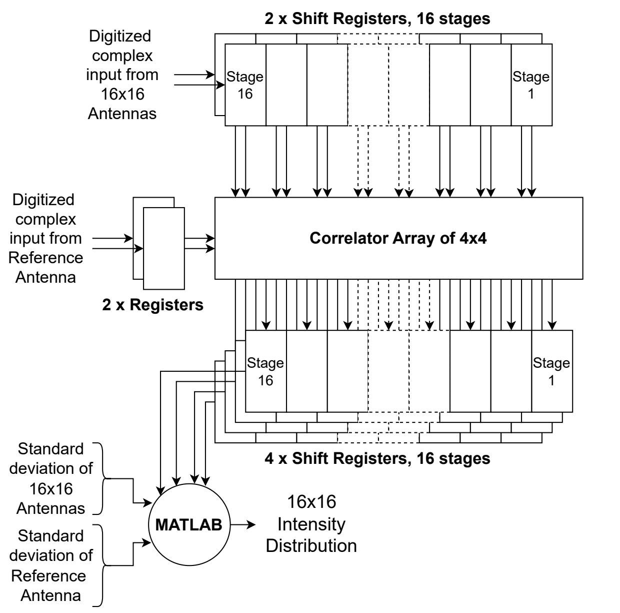

2 x Shift Registers, 16 stages
Digitized complex input from 16x16 Antennas
Stage 16
Stage 1
Digitized complex input from Reference Antenna
Correlator Array of 4x4
2 x Registers
Stage 16
Stage 1
4 x Shift Registers, 16 stages
Standard deviation of 16x16 Antennas
Standard deviation of Reference Antenna
MATLAB
16x16 Intensity Distribution

FIGURE 15: The proposed realization of the correlator on an FPGA for an antenna array of 16×16.

Transformation on the two-dimensional visibility component in Eq. 4, which is produced by the correlator MATLAB model. The behavioral MATLAB model of the RMs in Table 1 are incorporated. The output-quality performance of the conventional xRM behavior embedded on the correlator is presented in [1] and ignored here. Aside from these multipliers, an ideal multiplier with floating point accuracy is also included in the measurement to evaluate the accuracy of the xRM structure. The output-quality metrics are the SNR, the SFDR, and the RMS level as presented in [1]. An antenna array of 16×16 and an integration length of 64 is used for a manageable simulation time. The hardware and power evaluation on the FPGA was limited to a correlator array size of 4×4. Therefore, the output-quality evaluation of the architecture was completely conducted on the MATLAB behavioral model. The propose realization of the correlator on a FPGA for an antenna array size of 16×16is illustrated in Fig. 15. A single source is assumed to be located by the antenna array at $\mathcal{I}(\frac{1}{8},\frac{5}{8})$. The antennas are assumed to be spaced by $\frac{1}{2}\lambda$. For these metrics, Monte Carlo simulations have been done using the same setup of the latter with SNR input range from -20 dB to +44 dB where the results are averaged over 25 runs.

Fig. 16 illustrates the behavior of the correlator with an input SNR up to 10 dB. All the RM structures give similar performance, where an increase of SNR_{in} leads to an increase of SNR_{dB}. Between 10 dB and 30 dB, the curves of all the xRM structures deviate from the ideal curve by approximately 5 dB. [1] had already demonstrated the behavior of the correlator when using the xRM structure with the ISH1 configuration, where for a SNR_{in} larger than 10dB, the correlator output deviates from the ideal curve, as shown in Fig. 16. It can be observed that the xRM with the $\text{FPGA}_{\text{ISH1}}$ combination have a similar deviation curve as the ISH1 combination when SNR_{in} is beyond 10 dB. When SNR_{in} reaches 30 dB or higher, then the quantization noise present on the inputs of the correlator will also be fed in with the pairs of antennas. This occurs when the RMS level of the noise added with the input signal is larger than the quantization step size [1]. With the 5σ-clipping set on the AGC, the quantization noise is uncorrelated up to approximately 30 dB. Fig. 16 illustrates the curve of the accurate multiplier deviates from the ideal case for SNR_{in} beyond 30 dB. This holds for most of the FPGA_ISH types but with a faster pace compared to the accurate multiplier. the quantization noise between different antennas will be correlated for SNR_{in} larger than 30 dB with 5σ-clipping settings on the AGC, which eliminates the linear rise of the SNR_{dB} with SNR_{in}. The latter holds for the rest of the RM combination structure, as their curve deviates from the ideal curve when SNR_{in} is larger than 30 dB.

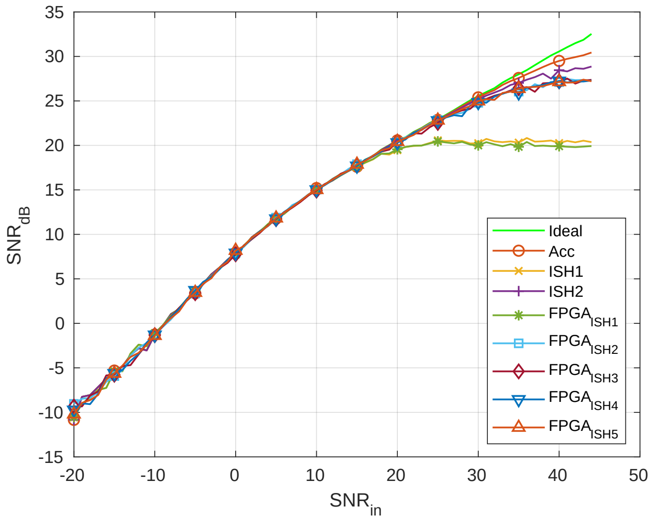


FIGURE 16: SNR within correlator architecture's output map, that is based on a 16×16 antenna array with a single point source as a function of the SNR at the input of each antenna.

Fig. 17 presents the result of the SFDR_{dB} as a function of SNR_{in}. For SNR_{in} less than 10 dB, the SFDR_{dB} of all the multipliers have a similar performance. The xRM structure with the $\text{FPGA}_{\text{ISH1}}$ combination has a similar deviation from the ideal curve as the ISH1 combination, which is at SNR_{in} larger than 10 dB. Beyond 20 dB of SNR_{dB}, the rest of the xRM structure deviates from the ideal curve.

Fig. 18 presents the RMS level of the noise within the resulted map of the correlator architecture as a function of SNR_{in}. Similar [1], the R_{dB} for all RM structures are not affected by the noise of the antenna signals up to -10 dB, while a reduction does occur when SNR_{in} exceed -10 dB. Increasing SNR_{in} beyond 10 dB does not lower the R_{dB} for

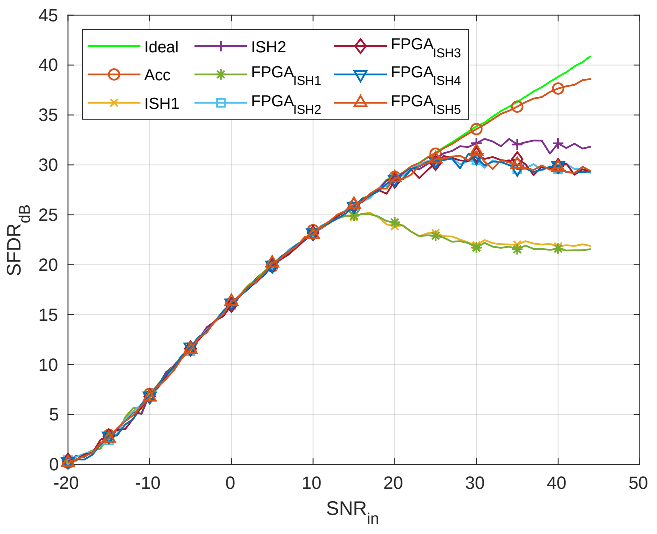


FIGURE 17: SFDR within the correlator architecture output map, that is based on a 16×16 antenna array with a single point source as a function of the SNR at the input of each antenna.

ISH1 and $FPGA_{ISH1}$ xRMs. For both cases, the quantization noise is correlated at a higher R_{dB} compared to the rest of the RM structure. The rest of the xRM curves have a deviation from the ideal curve beyond 25 dB. The FPGA-based xRM have a higher exponential deviation compared to the ASIC-based xRM. The accurate RM have the lowest deviation from the ideal curve at SNR_{dB} beyond 30 dB.

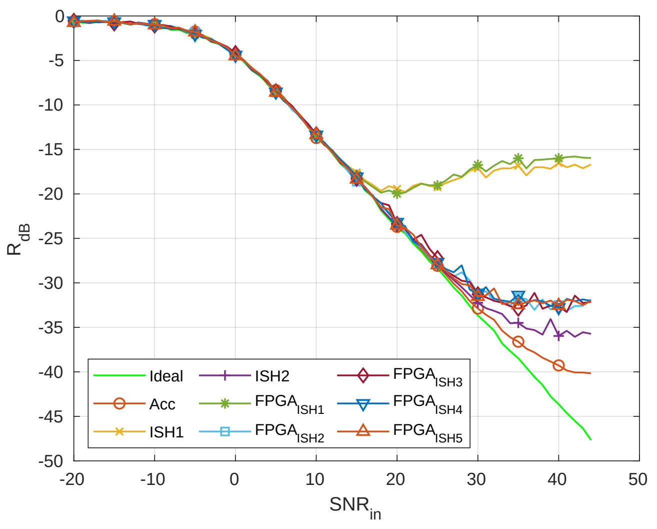


FIGURE 18: RMS of the noise within the correlator architecture output map, that is based on a 16×16 antenna array with a single point source as a function of the SNR at the input of each antenna.

VII. CONCLUSION

This paper has presented the hardware resources utilization, latency, and power of using approximate recursive multipliers in a correlator array architecture and deployed on an FPGA device with a four-input LUT base. These performances were evaluated for the correlator array of 1×1, 2×2, and 4×4. To answer the research question "What is the most optimal (complex) multiplier for energy-efficiency improvement on FPGA?", the approximate computing technique was used. The most optimal complex multiplier for energy-efficiency improvement on FPGA comes down to the performance of the approximate multipliers. A set of power Pareto-optimal 8-bit xRMs structures for constructing xMAC accelerators was incorporated in the correlator array architecture with the sizes of 1×1, 2×2, and 4×4, and deployed on Intel's Cyclone Clone IV E FPGA device to evaluate the number of LUTs savings, speedup, and power savings. From these results, the following can be concluded:

1) Regarding the number of LUTs savings, FPGA_ISH1 has the highest with an average of 6.27%.
2) Regarding speedup, the number of LUTs on the internal critical path of the FPGA by using any of the xRM structures does not alter and thus is not a dominant component for speeding up the circuit design.
3) Regarding power savings on core dynamic, FPGA_ISH5 has the highest with a 26.52% at the correlator array size of 1×1, and 2.9% at 2×2. FPGA_ISH1 has the highest with 1.61% at the correlator array size of 4×4. As the correlator array size increases from 1×1 to 2×2, the highest power reduction decreases by a factor of 9.
4) Regarding total power savings, the form of the power savings is almost similar to the core dynamic power savings. However, the magnitude of the power savings for all xRM structures is less than 1%. The reason is that the core static power of the FPGA consumes way more than the dynamic power.

An antenna array of 16×16 was used to evaluate the output-quality image produced from the correlator behavioral model embedded with the behavior models of the 8-bit input RM structures, that employ the ISH methodologies. From the results, the following can be concluded:

1) FPGA_ISH1 xRM structure embedded on the correlator behavior would have similar output-quality performance as the ISH1 xRM i.e., no effect would be introduced on synthesizing an image when the SNR at the input of the antennas are below 10 dB.
2) The rest of the FPGA-based xRMs structure used in this paper have a noise correlation at the same SNR_{in} level as the ISH2 with a more aggressive deviation from the ideal curve.

Unfortunately, the power savings result of the correlator array N×N the SIPO and PISO shift registers for an antenna array of 16×16 was not concluded. Another solution for this is to implement the correlator array of N×N to a set of the FPGA. Each FPGA would process a subset of antennas in the antenna array of 16×16. The number of devices, that are required is defined by the ratio of the antenna array size to the correlator array size.

The results indicate, that the most optimized correlator architecture for energy-efficiency improvement on an FPGA with a 4-input LUTs base (Cyclone Clone IV E) is at the size of 1×1 size and integration length of 64. With FPGA_ISH5 xRM structure incorporated, a power savings of 26.52% is achieved on the core dynamic power dissipation. However, from the total power savings results, all the approximate multipliers structures in this paper are less than 1%. The average number of LUTs savings for this realization is 4.2%. Additionally, most FPGA_ISH5 xRM structures do not have the worst deviation from the ideal curve on the correlator's output-quality metrics as the ISH1 and FPGA_ISH1.

VIII. FUTURE WORK

The correlator array implementation misses the extension to work with an antenna array larger than its size. To further investigate the approximate multiplier on a large telescope, multiple boards are required to measure the power reduction on a large scale.

To take advantage of using approximate computing on Intel's FPGA, regarding power savings and latency, the synthesis-tool method of mapping a circuit design from the VHDL to an FPGA must be considered. Otherwise, another solution will be to find the power Pareto-optimal approximate multipliers set for a given number of CMACs operators, which will raise the design-space for exploration.

To further investigate the energy-efficiency of the correlators on an FPGA, it would be of interest to investigate whether the approximate multipliers benefits of reducing power becomes relevant as the integration length rises. Thus, finding the threshold of the accumulator length before it becomes the power dominant and obliterates the benefits of applying approximate multipliers on FPGA. In such cases, approximate adders designed for FPGA could be of interest in the replacement of the ripple-carry-adder in the accumulator component, illustrated in Fig. 10, for further investigation.

Regarding the output-quality metrics performance of the correlator, the approximate 8-bit input recursive multipliers, that employ the ISH methodology to build xMAC accelerators, were analyzed. As indicated in the quality-output result and [1], as long the SNR at the input of the correlator is low, a more aggressive approximation could be applied, and thus higher power savings are achievable. The number of approximate multipliers for the output-quality evaluation can be expanded to the FPGA-based approximate multipliers constructed from the Xel-FPGA framework [31] and the Approximate Logic Synthesis (ALS) methodology [32].

REFERENCES

[1] Kokkeler, A.B.J., Gillani, G.A. and Boonstra, A.J. Modeling the effects of power efficient approximate multipliers in radio astronomy correlators. Exp Astron 57, 11 (2024). https://doi-org.ezproxy2.utwente.nl/10.1007/s10686-024-09921-3

[2] G. A. Gillani, M. A. Hanif, B. Verstoep, S. H. Gerez, M. Shafique and A. B. J. Kokkeler, "MACISH: Designing Approximate MAC Accelerators With Internal-Self-Healing," in IEEE Access, vol. 7, pp. 77142-77160, 2019, doi: 10.1109/ACCESS.2019.2920335.

[3] P. Kulkarni, P. Gupta, M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in Proc. of the VLSI Design IEEE, Chennai, India, 2011, pp. 346–351.

[4] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in Proc. of the ICCAD, Austin, TX, USA, 2016, pp. 1–8.

[5] G.A. Gillani, M.A. Hanif, M. Krone, S.H. Gerez, M. Shafique, and A.B.J. Kokkeler, "SquASH: Approximate Square-Accumulate with Self-Healing," IEEE Access, DOI 10.1109/ACCESS.2018.2868036, vol. 6, pp. 49112-49128, 2018.

[6] National Radio Astronomy Observatory, "What are radio telescopes?" [Online]. Available: https://public.nrao.edu/telescopes/radio-telescopes/

[7] A. R. Thompson , J. M. Moran , G. W. Swenson Jr., " Introduction and Historical Review," in Interferometry and Synthesis in Radio Astronomy, 3th ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Cham, 2017, ch. 1, sec. 1.3, pp. 13–43. [Online]. Available: https://doi.org/10.1007/978-3-319-44431-4

[8] A. R. Thompson , J. M. Moran , G. W. Swenson Jr., "Introductory Theory of Interferometry and Synthesis Imaging," in Interferometry and Synthesis in Radio Astronomy, 3th ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Cham, 2017, ch. 2, sec. 2.4, pp. 73–76. [Online]. Available: https://doi.org/10.1007/978-3-319-44431-4

[9] A. R. Thompson , J. M. Moran , G. W. Swenson Jr., "Analysis of the Interferometer Response," in Interferometry and Synthesis in Radio Astronomy, 3th ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Cham, 2017, ch. 3, sec. 3.1, pp. 89–95. [Online]. Available: https://doi.org/10.1007/978-3-319-44431-4

[10] A. R. Thompson , J. M. Moran , G. W. Swenson Jr., "Phased Arrays and Correlator Array," in Interferometry and Synthesis in Radio Astronomy, 3th ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Cham, 2017, ch. 5, sec. 5.3.1, pp. 162–164. [Online]. Available: https://doi.org/10.1007/978-3-319-44431-4

[11] A. R. Thompson , J. M. Moran , G. W. Swenson Jr., "Digital Signal Processing," in Interferometry and Synthesis in Radio Astronomy, 3th ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Cham, 2017, ch. 8, pp. 309–373. [Online]. Available: https://doi.org/10.1007/978-3-319-44431-4

[12] van der Veen, AJ., Wijnholds, S.J., Sardarabadi, A.M. (2019). Signal Processing for Radio Astronomy. In: Bhattacharyya, S., Deprettere, E., Leupers, R., Takala, J. (eds) Handbook of Signal Processing Systems. Springer, Cham. [Online]. Available: https://doi.org/10.1007/978-3-319-91734-4_9

[13] L. R. D'Addario, "Low-Power Correlator Architecture for the Mid-Frequency SKA", March 2011, SKA Memo 133.

[14] J. L. Hennessy , D. A. Patterson, Computer Architecture: A Quantitative Approach, 6th ed., Cambridge, MA 02139, USA: Morgan Kaufmann, 2019, pp. 2–64.

[15] R. Jongerius, S. Wijnholds, R. Nijboer and H. Corporaal, "An End-to-End Computing Model for the Square Kilometre Array," in Computer, vol. 47, no. 9, pp. 48-54, Sept. 2014, doi: 10.1109/MC.2014.235.

[16] M. van Haarlem, "Square Kilometre Array." [Online]. Available: https://www.astron.nl/telescopes/square-kilometre-array/ (accessed Sept. 26, 2023)

[17] Rob V. van Nieuwpoort and John W. Romein. 2009. "Using many-core hardware to correlate radio astronomy signals". In Proceedings of the 23rd international conference on Supercomputing (ICS '09). Association for Computing Machinery, New York, NY, USA, 440–449. https://doi.org/10.1145/1542275.1542337

[18] R. van Nieuwpoort and J. W. Romein, "Building Correlators with Many-Core Hardware," in IEEE Signal Processing Magazine, vol. 27, no. 2, pp. 108-117, March 2010, doi: 10.1109/MSP.2009.935385.

[19] John W. Romein, P. Chris Broekema, Jan David Mol, and Rob V. van Nieuwpoort. 2010. "The LOFAR correlator: implementation and performance analysis". SIGPLAN Not. 45, 5 (May 2010), 169–178. https://doi.org/10.1145/1837853.1693477

[20] Harvey R. Butcher "LOFAR: first of a new generation of radio telescopes", Proc. SPIE 5489, Ground-based Telescopes, (28 September 2004); https://doi.org/10.1117/12.548806

[21] Fiorin, L., Vermij, E., van Lunteren, J. et al. Exploring the Design Space of an Energy-Efficient Accelerator for the SKA1-Low Central Signal Processor. Int J Parallel Prog 44, 1003–1027 (2016). https://doi.org/10.1007/s10766-016-0420-y

[22] L. de Souza, J. D. Bunton, D. Campbell-Wilson, R. J. Cappallo and B. Kincaid, "A Radio Astronomy Correlator Optimized for the Xilinx Virtex-

4 SX FPGA," 2007 International Conference on Field Programmable Logic and Applications, Amsterdam, Netherlands, 2007, pp. 62-67, doi: 10.1109/FPL.2007.4380626.

[23] W. Kamp, N. Abel and G. Comoretto, "Complex Multiply Accumulate Cells for the Square Kilometre Array Correlators," 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 2018, pp. 1-6, doi: 10.1109/RECONFIG.2018.8641708.

[24] L. Gerlach, G. Payá-Vayá and H. Blume, "An area efficient real- and complex-valued multiply-accumulate SIMD unit for digital signal processors," 2015 IEEE Workshop on Signal Processing Systems (SiPS), Hangzhou, China, 2015, pp. 1-6, doi: 10.1109/SiPS.2015.7345019.

[25] S. Guo, L. Zheng and X. Jin, "Accelerating a radio astronomy correlator on FPGA," 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon, Korea (South), 2018, pp. 85-89, doi: 10.23919/ICACT.2018.8323654.

[26] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu and J. Han, "Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications," in Proceedings of the IEEE, vol. 108, no. 12, pp. 2108-2135, Dec. 2020, doi: 10.1109/JPROC.2020.3006451.

[27] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. 2018. "SMApproxlib: library of FPGA-based approximate multipliers," In Proceedings of the 55th Annual Design Automation Conference (DAC '18). ACM, New York, NY, USA, Article 157, 6 pages. doi: https://doi.org/10.1145/3195970.3196115

[28] S. Yao and L. Zhang, "Hardware-Efficient FPGA-Based Approximate Multipliers for Error-Tolerant Computing," 2022 International Conference on Field-Programmable Technology (ICFPT), Hong Kong, 2022, pp. 1-8, doi: 10.1109/ICFPT56656.2022.9974399.

[29] R. van Loo, "Investigating Approximate FPGA multiplication for increased power-efficiency", M.S. Thesis Faculty EEMCS, Univ. Of Twente, Enschede, The Netherlands, Nov. 2022. [Online]. Available: https://purl.utwente.nl/essays/93756

[30] R.H. van der Wijk, "Designing 8-bit approximate multipliers for FPGA using internal self-healing", B.S. Thesis, Faculty EEMCS, Univ. Of Twente, Enschede, The Netherlands. [Online]. Available: https://purl.utwente.nl/essays/94789

[31] B. S. Prabakaran, V. Mraze, Z. Vasicek, L. Sekanina and M. Shafique, "Xel-FPGAs: An End-to-End Automated Exploration Framework for Approximate Accelerators in FPGA-Based Systems", International Conference on Computer-Aided Design (ICCAD), San Francisco, CA, USA, Nov. 2023, doi: https://doi.org/10.48550/arXiv.2303.04734

[32] Xiang, Z., Liu, N., Yao, Y., Yang, F., Zhuo, C., Qian, W. (2022). Approximate Logic Synthesis for FPGA by Decomposition. In: Liu, W., Lombardi, F. (eds) Approximate Computing. Springer, Cham. https://doi.org/10.1007/978-3-030-98347-5_7

[33] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," in Proceedings of the IEEE, vol. 83, no. 4, pp. 498-523, April 1995, doi: 10.1109/5.371964.

[34] Amara Amara, Frédéric Amiel and Thomas Ea, "FPGA vs. ASIC for low power applications", Microelectronics Journal, vol. 37, no. 8, pp. 669–677, 2006, https://doi.org/10.1016/j.mejo.2005.11.003.

[35] L. Luet Ng, K. Ho Yeap, M. Wan Ching Goh, and V. Dakulagi, 'Power Consumption in CMOS Circuits', Electromagnetic Field in Advancing Science and Technology. IntechOpen, Mar. 29, 2023. doi: 10.5772/intechopen.105717.

[36] Iida, M. (2018). What Is an FPGA?. In: Amano, H. (eds) Principles and Structures of FPGAs. Springer, Singapore. https://doi.org/10.1007/978-981-13-0824-6_2

[37] Amagasaki, M., Shibata, Y. (2018). FPGA Structure. In: Amano, H. (eds) Principles and Structures of FPGAs. Springer, Singapore. https://doi.org/10.1007/978-981-13-0824-6_3

[38] Greg Stitt, "FPGA Timing Optimization: Background and Challenges", Intel DevCloud Training Module, University of Florida. [Online]. Available: https://github.com/ARC-Lab-UF/intel-training-modules/blob/master/timing/timing_background.pptx

[39] Intel Incorportions, "Intel® Quartus® Prime Standard Edition User Guide: Timing Analyzer", v18.1, 2018. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683068/18-1/timing-analysis-introduction.html

[40] Intel Incorporations, "Intel® Quartus® Prime Standard Edition User Guide: Third-party Simulation", v22.1, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683080/22-1/simulation-levels.html

[41] A. Mishchenko, et al. ABC: A system for sequential synthesis and verification. 2007 [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/abc/

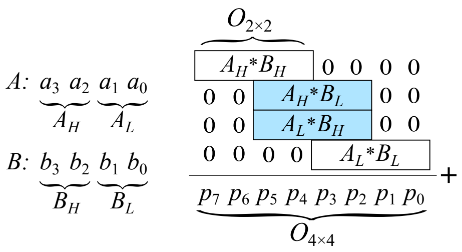
[42] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

[43] L. Amarú, P. E. Gaillardon, G. De Micheli, "The EPFL Combinational Benchmark Suite", In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, CA, USA, 2015.

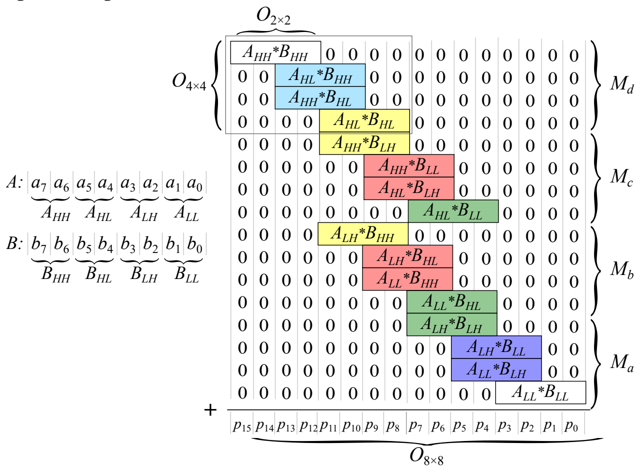
.

APPENDIX A RECURSIVE MULTIPLIERS STRUCTURE

An n-bit recursive multiplier (RM) can be designed using $(\frac{n}{2})^2$ elementary 2-bit multipliers, where n represent the input bit width of the multiplier, $n \in \{4,8,16,32,...\}$. The elementary 2-bit multipliers generate the partial products of the multiplication operation. The summation of these partial products with bit-shift is the resultant output of an n-bit RM. As an example, Fig. 19 illustrates a 4-bit RM and 8-bit RM structure that compose four and sixteen 2-bit elementary multipliers, respectively. To transform the RM to an approximate



(a) 4-bit input recursive multiplication (RM) requires four 2-bit input multipliers.



(b) 8-bit input recursive multiplier (RM) requires sixteen 2-bit input multipliers.

FIGURE 19: Recursive n-bit multiplication utilizes elementary 2-bit input multipliers. The same colors show equal numerical weight 2-bit multipliers that can be approximated with $+\delta$ and $-\delta$ errors to enable Internal-Self-Healing (ISH) [2].

RM (xRM), either the adders, the 2-bit multipliers, or a combination of both must be approximated. The proposed xRM in [2] utilized a combination of several approximate 2-bit multipliers ASIC-based designs, where the best combination for reduction of power or area is selected based on the input distribution and the output quality constraints. The truth-table of these 2-bit multiplier types is shown in Fig. 20.

APPENDIX B FPGA CORE STRUCTURE

In principle, the FPGA architecture consists of three programmable logic blocks, namely the logic element (LE), the I/O element, and the routing element, that is placed in an island-style structure [36] [37]. The LE expresses a logic function, the I/O element provides an external interface, and the routing element interconnects different blocks. Aside from these logic blocks, FPGAs also contain numerous hardened blocks for common specialized tasks, such as the DSP units, the embedded memory to increase the calculation ability, and the phase-locked loop (PLL) to provide a clock network within the FPGA. The logic blocks are named differently among FPGA vendors. For Xilinx FPGA devices, it is named Configurable Logic Block (CLB) and for Intel FPGA, it is named Logic Array Block (LAB). A logic tile in the FPGA structure is a set of neighboring LE, and two routing elements (the connection block and the switch block). The internal structure of the LE consists of a LUT, a flip-flop (FF), and a selector. The LUT with k-inputs can express a logic function with a k-input truth table. The FF stores the output value from the LUT. The selector controls whether the value of the LUT or the one stored in the FF is outputted from the LE. The area efficiency in the LE depends on how efficiently the internal component is used when a circuit is implemented on the FPGA. When the LUT is limited by z-input, due to the 2^z configuration memory bit for the LE, the area efficiency of the internal LE increases. However, in cases where a k-input logic function is greater than the z-input LUT, the total number of LE also increases in order to map completely the k-input logic function on the FPGA. When the number of I/O pins increases, the routing part increases, and thus increases the area per logic tile. The total area of a circuit design implemented on an FPGA is determined by the product of the total number of logic blocks used with the area per logic tile. Modern FPGAs also employ the adaptive LUTs to gain a higher area efficiency, since their internal structure is capable of implementing more logic functions compared to LE [37]. The naming convention of the adaptive LUTs for Intel and Xilinx are the adaptive logic module (ALM) and adaptive CLB, respectively.

APPENDIX C FPGA POWER CONSUMPTION

Whenever the semiconductor technology of the FPGA is constructed via a Static Random Access Memory (SRAM) base, then the power profile of the FPGA is similar to CMOS-technology [34] [36]. In CMOS circuit, the total power consumption is the sum of the dynamic and the static power components within the circuit. The dynamic power component is referred to as the power consumption of the circuit during a switching activity, whereas the static power component is the power lost produced from the current flow leakage through inactive transistors in the circuit [35]. This leakage is the sub-threshold current leakage and the gate current leakage which is predominantly caused by the short-channel effects of the transistor physical size. As the transistor technology continues to shrink, the sub-threshold leakage current increases exponentially. The static power component is expressed as,

$$P_{static} = V_{dd} \cdot I_{leakage} \tag{28}$$

	Accurate (M)				M1				M2				M3				M4			
C / A	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	0	1	2	3	0	0	2	2	0	1	2	3	0	1	2	3
2	0	2	4	6	0	2	4	6	0	2	4	6	0	2	4	6	0	2	4	6
3	0	3	6	9	0	3	6	7	0	2	6	9	0	3	6	11	0	3	6	5

FIGURE 20: Truth-tables of 2-bit multipliers. A and C are 2-bit inputs having a range of 0 to 3, shown in decimal numbers. Accurate (M) has all the products correct. M1 has only one output approximated ($3\times3\mapsto7$ instead of 9) that produces an error of -2 [3]. M2 has three approximated outputs [4], wherein each approximate produces an error of -1 ($1\times1\mapsto0$ instead of 1). M3 is similar to M1, however, it produces an error of +2 to complement M1 [5]. M4 produces an error of -4 while approximating the product (output) for the same combination of inputs, i.e., $A = 3$ and $C = 3$ [2]. The figure is from [1].

where $I_{leakage}$ denotes the total leakage current in the transistor. Dynamic power dissipation on the other hand is dominated by the transition of logic state, either from logic 0 to logic 1 or vice versa, at the circuit nodes. Moreover, it is also influenced by a concurrent transition of both PMOS and NMOS transistors on a node. This concurrent transition would cause a temporary short-circuit at a short period, thus resulting in a power dissipation (P_{short}) in the CMOS circuity. However, this short-circuit power dissipation is considerably lower than the power switching dissipation with high switching activities. In terms of power saving, the interest is in finding techniques that reduce power by lowering the switching activity. For completeness on the dynamic power dissipation in a CMOS circuit, its expression is given by,

$$P_{dynamic} = P_{switch} + P_{short} \tag{29}$$

$$P_{switch} = \alpha_{0\mapsto1} f_{clk} C_L V_{dd}^2 \tag{30}$$

$$P_{short} = T_{sc} V_{dd} I_{peak} \tag{31}$$

where $\alpha_{0\mapsto1}$ is the activity factor, f_{clk} is the clock frequency the CMOS circuit is operating, C_L is the load capacitance which charges or discharges when a transition occurs, V_{dd} is the supply voltage of the CMOS cirtuity, T_{sc} is the rising edge or falling edge time of the input signal and I_{peak} is the peak current when a short-circuit occurs [33]–[35]. The activity factor is defined as the probability a circuit node changes from logic 0 to logic 1 within one clock cycle. When a signal is triggered continuously on every clock cycle, e.g. the clock signal itself, then the activity factor on that node is equal to 1. Thus, $P_{switch} = f_{clk} C_L V_{dd}^2$ on that node. On the other hand, when a signal is toggled continuously every two clock cycles, the activity factor is still equal to 1 by definition. Thus, the $P_{switch} = f_{clk} C_L V_{dd}^2$. In cases where the switching occurs irregularly on the node with a Uniform Distribution, then the activity factor is determined by multiplying the probability the node switches from logic 0 (P(Y=1)), with the probability it switches to logic 1 (P(Y=1)) [35],

$$\alpha_{0\mapsto1} = P(\text{Y=0}) \cdot P(\text{Y=1}) \tag{32}$$

For example, in the activity factor for the elementary 2-bit input accurate multiplier module in [2], the MSB node produces logic 1 only once for all 16 input combinations. Hence $P(\text{Y=1}) = \frac{1}{16}$ and $P(\text{Y=0}) = \frac{15}{16}$, thus the probability the node is active is approximately to 0.06.

To define the dynamic power model for SRAM-based FPGA, the dynamic power model for all hardware resources, that cause a transition of one logic state to another, must be included. Since power switching is the dominant component of the dynamic power model, the short-circuit power dissipation would be ignored. Thus, the expression of the dynamic power of the FPGA is given as,

$$P_{dynamic} = f_{clk} V_{dd}^2 \sum_{k=1}^{K} \alpha_{0\mapsto1_k} C_k U_k \tag{33}$$

where f_{clk} and V_{dd} are the operating frequency and the supply voltage of the FPGA, respectively. $\alpha_{0\mapsto1_k}$, C_k and U_k are respectively the activity factor, the effective capacitance, and the utilization of kth LUT [34].

APPENDIX D SIGNAL FLOW GRAPH DERIVATION

The derivation of an architecture from an algorithm in this work is based on deriving the Signal Flow Graph (SFG). The idea of doing so is to demonstrate the hardware structure that is needed for executing the algorithm. In practice, the algorithm descriptions refer to either a programming language or algebraic descriptions. In this section, the SFG derivation is illustrated from a simple algebraic description for demonstration purposes. The information in this appendix is partially collected from a course offered at the University of Twente, namely *Computer Architecture 2*, course module code: *192130250*.

A. DESIGN FLOW OF SFG

The design flow of the SFG consists of the following steps:

1) Idea
2) algebraic description or program (imperative)
3) single assignment (function)
4) recurrent relations
5) dependency graph
6) signal flow graph

The algebraic description for this demonstration will be the matrix-vector-multiplication expressed as,

$$c_i = \sum_{j=0}^{N-1} a_{i,j} \cdot b_j \tag{34}$$

where *i* and *N* is an arbitrary natural number (ℕ).

B. ALGORITHM TO DEPENDENCY GRAPH

The first objective is to derive the dependency graph (DG) from the algebraic description. DG consists of vertices and edges. The vertices are expressed as operations and the edges as the data flows in the graph. The graph describes the dependencies of the data between the vertices. The data through an edge of the DG is a scalar value. The vertices are a function of a scalar value. The internal details of the vertices are determined at the single assignment phase. With an associative operation in the algorithm, the single assignment is the function occurring repeatably in the algorithm. For Eq. 34, the vertices are a combination of multiplication and addition operations. The recurrent relation of the algorithm establishes the dependencies of the vertices. With an associative operation in the algorithm description, two recurrent relations can be derived as illustrated in Fig. 21. For a complete description, the recurrent relation alternative 1 is expressed as,

$$s_{i,j} = s_{i,j-1} + a_{i,j} \cdot b_j \tag{35}$$

with a minimum bound of $s_{i,\text{-}1} = 0$ and maximum bound of $c_i = s_{i,N-1}$, whereas the recurrent relation alternative 2 is expressed as,

$$s_{i,j} = s_{i,j+1} + a_{i,j} \cdot b_j \tag{36}$$

with the minimum bound of $c_i = s_{i,0}$ and the maximum bound of $s_{i,N} = 0$. For both recurrent relations alternatives, $i \in \{0, 1, \dots, K\}$ and $j \in \{0, 1, \dots, N-1\}$.

$$c_i = \underbrace{\underbrace{a_{i,0}.b_0 + ...a_{i,j-1}.b_{j-1}}_{s_{i,j-1}} + a_{i,j}.b_j}_{s_{i,j} = s_{i,j-1} + a_{i,j}.b_j} + a_{i,j+1}.b_{j+1} + ...a_{N-1}.b_{N-1}$$

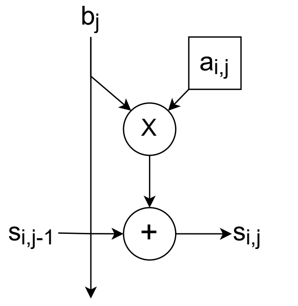
(a) Recurrent relation alternative 1.

$$c_i = a_{i,0}.b_0 + ...a_{i,j-1}.b_{j-1} + \underbrace{a_{i,j}.b_j + \underbrace{a_{i,j+1}.b_{j+1} + ...a_{N-1}.b_{N-1}}_{s_{i,j+1}}}_{s_{i,j} = s_{i,j+1} + a_{i,j}.b_j}$$

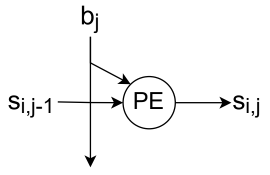
(b) Recurrent relation alternative 2.

FIGURE 21: Recurrent relations derived from Eq. 34.

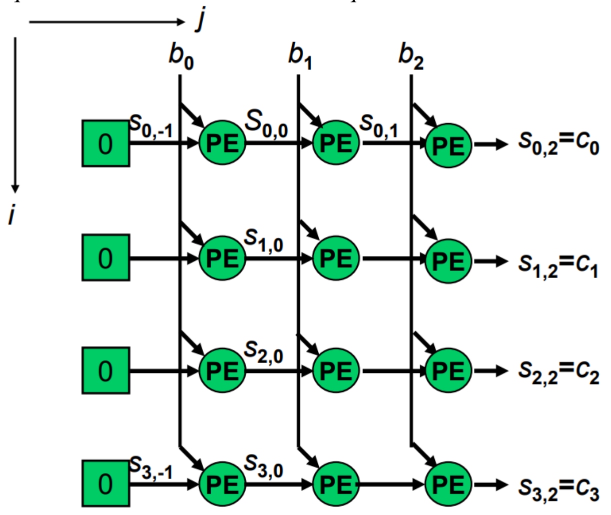
Each recurrent relation represents a DG. For example, Fig. 23a and Fig. 23b illustrate the internal structure of the vertices, and its entry and output, respectively, from the recurrent relation alternative 1 (see Eq. 35). The number of indices in the recurrent relation defines the dimension size of the DG. With the two indices in the recurrent relation alternative 1, a two-dimensional DG is generated. Fig. 22c illustrates this for K set to four and N to three. The DG from the recurrent



(a) The internal structure of the vertex for the recurrent relation in Eq. 35.



(b) The entry and output of the vertex for the recurrent relation in Eq. 35.



(c) Globally recursive graph.

FIGURE 22: The dependency graph from Eq. 35 with $K = 4$ and $N = 3$. A vertex is called a processing element (PE).

relation alternative 2 is a vertical mirrored version of the DG in Fig. 22, where all the horizontal edges are pointing to the left, the initial points are placed to the right, and the outputs c_i to the left.

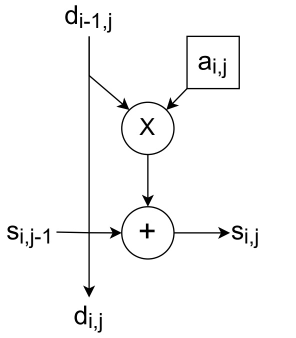
Both alternatives generate a DG with global dependencies on $b_0, b_1, \dots, b_{N-1}$. The generated DG from both alternatives is known as a globally recursive graph. This is due to an existent set of global data dependencies in the graph. A global data DG is limited to be manipulated and transformed into an SFG. To overcome this, the global data dependencies are transformed into local data dependencies by adjusting the recurrent relations. For this transformation, the recurrent relation alternative 1, Eq 35, can be modified either as,

$$s_{i,j} = s_{i,j-1} + a_{i,j} \cdot d_{i-1,j} \tag{37}$$

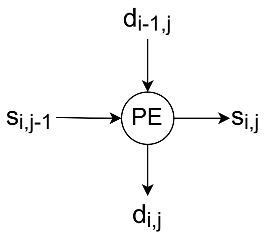
with $d_{i,j} = d_{i-1,j}$ and $d_{\text{-}1,j} = b_j$, or as

$$s_{i,j} = s_{i,j-1} + a_{i,j} \cdot d_{i+1,j} \tag{38}$$

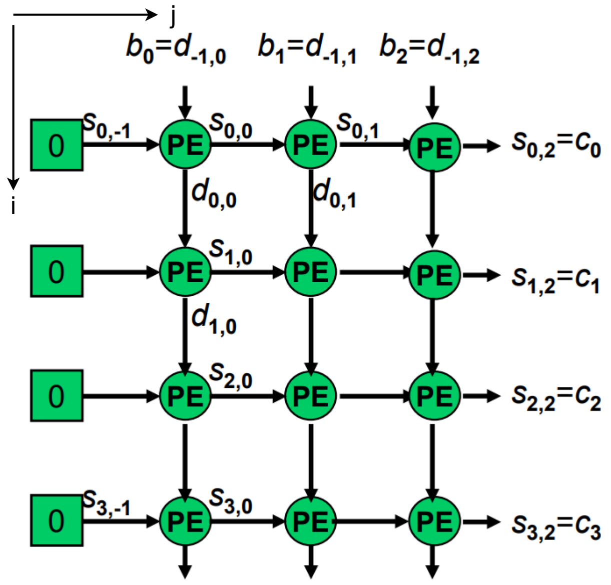
with $d_{i,j} = d_{i+1,j}$ and $d_{N,j} = b_j$. From this manipulation tactic, two extra recurrent relation alternatives are produced from Eq. 35, hence two local DGs can be generated. The resultant local DG from Eq. 37 is shown in Fig. 23. The local DG from Eq. 38 is a horizontally mirrored version of Fig. 23, where all the vertical edges are pointing upwards. Note that applying the same tactic on the recurrent relation alternative 2, two extra recurrent relation alternative are also produces, which in total four local DGs can be produced from Eq. 34.



(a) The graphical representation of the recurrent relation in Eq. 37.



(b) The single assignment process representation of the recurrent relation in Eq. 37.

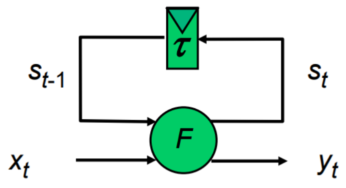


(c) Locally recursive graph.

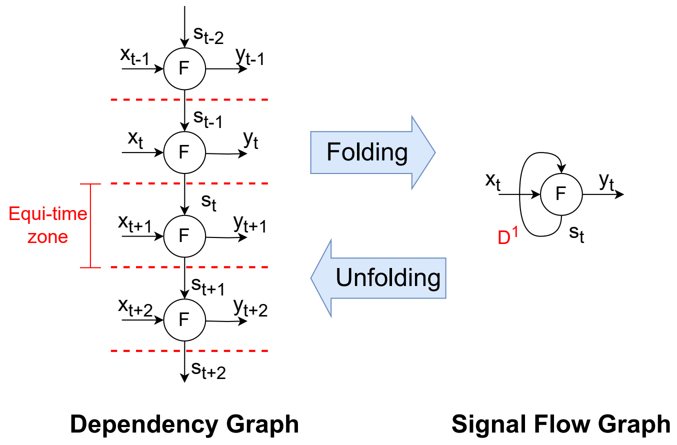
FIGURE 23: The dependency graph from Eq. 37 with $K = 4$ and $N = 3$.

C. DEPENDENCY GRAPH TO SIGNAL FLOW GRAPH

The SFG is drawn similarly to the DG with a different context on vertices and the data flows through the edges. In SFG, the data through an edge is described as streams (function on time), and the vertex is described as a function on streams. The vertices can be modeled as a synchronous machine, such as the Mealy- or the Moore Machine. These machines consist of registers and combination logic gate circuits. The registers are modeled by delay (D) units and the combinations of logic gate circuits by functions. Because of this, an SFG may contain loops as long as there is at least one unit delay in the loop. For this demonstration purpose, the mealy machine is considered. The algebraic function of the mealy machine has two inputs and two outputs, which is written as $y_t, s_t = F_{\text{mealy}}(x_t, s_{t\text{-}1})$. To create a loop edge in the SFG, a register is placed between the entry $s_{t\text{-}1}$ and the output s_t (see Fig. 24a). Since there is a repetitive execution of an operation (the repetitive vertices) in a DG, each execution can operate at *equi-time* zones. Mapping this repetitive execution along one axis folds the repetitive vertices of the DG to one SFG vertex (see Fig. 24b). Essentially, the SFG is a fold structure



(a) The graphical representation of the synchronous machine.



(b) Time-folding a set of re-instantiated nodes from a Dependency Graph to a node of a Signal Flow Graph.

FIGURE 24: A simple example of mapping the Dependency Graph to a Signal Flow Graph.

of the DG. Hence, a M-dimensional DG is mapped to a (M-1)-dimensional SFG. The data consumed and produced by the vertices in the M-dimensional DG within *equi-time* zones are scheduled in the (M-1)-dimensional SFG edges. The mapping of vertices and edges of a DG to an SFG can be done in different order, which derives different structures of the SFG. To clarify the mapping procedure mathematically, both the DG and the SFG are fully described by:

1) The set of vertices $V_n = \{\mathbf{v}_{\text{node}}\}$
2) The set of intermediate edges $E_{\text{int}} = \{(\mathbf{v}_{\text{source}}, \mathbf{e}_d)\}$
3) The set of input edges $E_{\text{in}} = \{(\mathbf{v}_{\text{in}}, \mathbf{e}_d)\}$
4) The set of output edges $E_{\text{out}} = \{(\mathbf{v}_{\text{out}}, \mathbf{e}_d)\}$

where $\mathbf{e}_d$ is the displacement vector between the source ($\mathbf{v}_{\text{source}}$) and the destination vertices ($\mathbf{v}_{\text{dest}}$), which it is defined as $\mathbf{e}_\mathrm{d}$ = $\mathbf{v}_\mathrm{dest}$ - $\mathbf{v}_\mathrm{source}$. The mapping from an M-dimensional DG to a (M-1)-dimensional SFG is described by,

$$\mathbf{w} = \mathbf{P} \cdot \mathbf{v} \tag{39}$$

where **P** is the processor assignment matrix, **w** is any vector of the SFG description, and **v** is the vector in the DG description. The processor assignment matrix is determined by,

$$\mathbf{P} = \mathbf{M}_T \cdot \mathbf{A} \tag{40}$$

$$= \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \cdot \mathbf{A} \tag{41}$$

where the mapping matrix $\mathbf{M}_\mathrm{T}$ is (M-1)×M identity matrix and **A** is a non-singular matrix. Because of this, **P** consists of (M-1) independent row vectors that span an (M-1)-dimensional hyperplane through an origin. Such a hyperplane can also be described by a vector **d** perpendicular to the hyperplane, in which the vector **d** is defined by $\mathbf{P} \cdot \mathbf{d}$ = 0. This vector **d** is known as the projection vector. Fig 25 illustrates three resulting SFG derivations from the local DG in Fig. 23 by three different processor assignment matrices **P**. These processor assignment matrices are equal to the null-space of their corresponding projection vector **d**.

The order in which data are consumed and produced in time by the SFG vertices is assigned by the scheduling (**S**). The scheduling of producing and consuming data, e.g. $y^{(t)}$, from the vertices on the SFG is determined by,

$$t = \mathbf{S} \cdot \mathbf{v} \tag{42}$$

where t indicates the time instance, that the data y from the vertex in **v**, will take place through the resulting SFG edge. The scheduling is determined by,

$$\mathbf{S} = \mathbf{S}_T \cdot \mathbf{A} \tag{43}$$

$$= \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix} \cdot \mathbf{A} \tag{44}$$

where the scheduling matrix $\mathbf{S}_\mathrm{T}$ is a 1×M identity matrix and A is a non-singular matrix. Because of this, the matrix **S** is also referred to as $\mathbf{s}^T$, which is a transpose of the scheduling vector **s**. With a scheduling vector, two vertices in a DG, e.g. $\mathbf{v}_x$ and $\mathbf{v}_y$, are scheduled at the same time if,

$$\mathbf{s}^T \cdot \mathbf{v}_x = \mathbf{s}^T \cdot \mathbf{v}_y \tag{45}$$

$$\mathbf{s}^T \cdot (\mathbf{v}_x - \mathbf{v}_y) = 0 \tag{46}$$

In other words, vertices in the DG, that are in the same hyperplane perpendicular to $\mathbf{s}^T$, are scheduled for the same time.

The restriction on applying the scheduling vector **s** and the projection vector **d** is that they cannot be perpendicular to each other for a non-singular matrix of $\mathbf{s}^T$ attached on top of **P**. Note that attaching the scheduling matrix $\mathbf{S}_\mathrm{T}$ on top of the mapping matrix $\mathbf{M}_\mathrm{T}$ forms the M×M identity matrix. Because of this, attaching the scheduling **S** matrix on top of the processor assignment matrix **P** becomes equivalent to the non-singular matrix **A**, and thus,

$$\det \begin{pmatrix} \mathbf{S} \\ \mathbf{P} \end{pmatrix} \neq 0 \iff \det(\mathbf{A}) \neq 0 \tag{47}$$

From this result, and $\mathbf{S} \approx \mathbf{s}^T$, and the processor assignment matrix **P** being equivalent to the null-space of the projection vector **d**, e.i. **P**·**d** = 0, the following statement holds,

$$\mathbf{s}^T \cdot \mathbf{d} \neq 0 \iff \det(\mathbf{R}) \neq 0 \tag{48}$$

where **R** is the matrix that consists of the matrix $\mathbf{s}^T$ attached on top of the matrix **P**.

Proof:

Any vector that is a linear combination of the row-vectors of **P** is perpendicular to **d**. The scheduling vector $\mathbf{s}^T$ is not perpendicular to **d**, hence it cannot be written as a linear combination of row-vectors of **P**. Therefore, **R** consist of a set of independent row-vectors, and thus

$$\mathbf{s}^T \cdot \mathbf{d} \neq 0 \implies \det(\mathbf{R}) \neq 0 \tag{49}$$

Suppose now that $\mathbf{s}^T \cdot \mathbf{d}$ = 0. Since **P**·**d** = 0 and $\mathbf{d} \neq 0$ implies that **R**·**d** = 0, and thus det(**R**) = 0. This means,

$$\mathbf{s}^T \cdot \mathbf{d} = 0 \implies \det(\mathbf{R}) = 0 \tag{50}$$

and thus its inverse also holds,

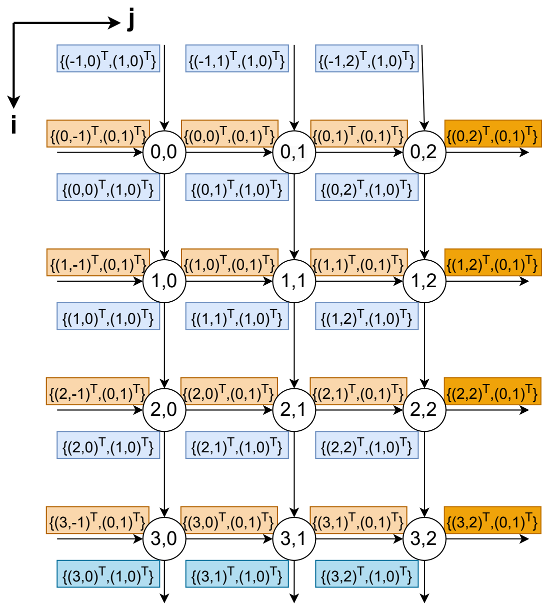
$$\det(\mathbf{R}) \neq 0 \implies \mathbf{s}^T \cdot \mathbf{d} \neq 0 \tag{51}$$

Combining the statement in Eq. 49 and Eq. 51, the statement in Eq. 48 is proven.

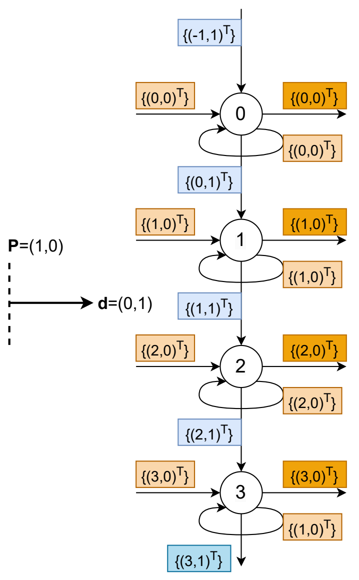
Furthermore, the scheduling vector $\mathbf{s}^T$ is also restricted by the direction of the edges in the DG. Since the edges in the DG express dependencies between the vertices, the scheduling of producing and consuming data between the vertex of the SFG must be consistent with the vertex of the DG. The scheduling vector $\mathbf{s}^T$ expresses the order in which the operations will be executed in the vertex of the SFG. An operation may not depend on an operation in the future. Therefore, the angle between the edges in the DG and the scheduling vector **s** must be less or equal to 90 degrees. Therefore, any dependency edge $\mathbf{e}_\mathrm{d}$ in the DG must hold the following,

$$\mathbf{s}^T \cdot \mathbf{e}_d \geq 0 \tag{52}$$

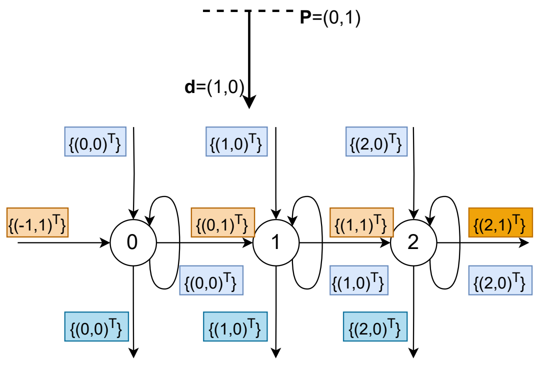
For instance, the local DG in Fig. 23 contains edges that are displaced either as $\mathbf{e}_\mathrm{d}$ = (0,1) or as $\mathbf{e}_\mathrm{d}$ = (1,0). With these two types of edges, the scheduling vector **s** is restricted clockwise from (0,1) to (1,0). With this restriction of the edges in combination with the projection vector **d** set to (0,1) to form the SFG in Fig. 25b), the scheduling vector is further restricted clockwise from (0,1) to (1,1). This is due the subscript i and j being natural numbers. With the projection vector **d** set to (1,0) to form the SFG in Fig. Fig. 25c, the scheduling vector restriction from the edges is further restricted clockwise from (1,1) to (1,0). With the projection vector **d** set to (1,1) to form the SFG in Fig. Fig. 25d, the scheduling vector stays restricted clockwise from (0,1) to (1,1). Fig 26 illustrates the



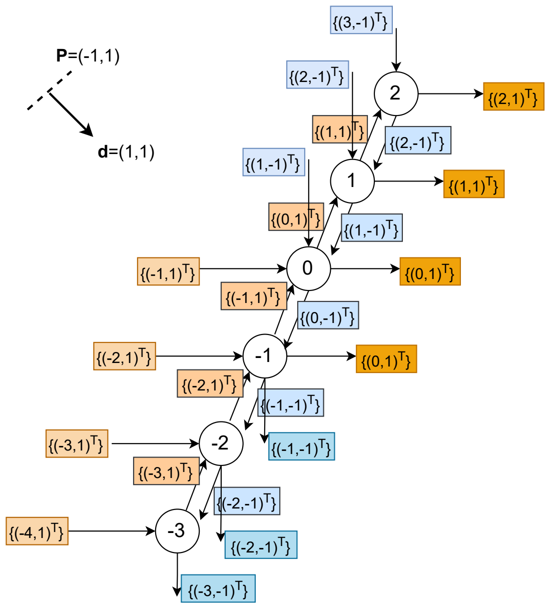
(a) The mathematical description of all vertices and edges of the Dependency Graph in Fig. 23.



(b) The resulting SFG for the projection vector **d** set to (0,1), e.i. an increase in subscript j.



(c) The resulting SFG for the projection vector **d** set to (1,0), e.i. an increase in subscript i.



(d) The resulting SFG for the projection vector **d** set to (1,1), e.i. an increase in both subscripts.

FIGURE 25: Mapping the Dependency Graph to Signal Flow Graphs by using three different projection vectors **d** set in (i,j) form. The processor assignment matrix **P** being equivalent to the null-space of the projection vector **d** and using the mapping description in Eq. 39, three different Signal Flow Graph structures are derived from the Dependency Graph. Note the superscript (T) indicates the transpose transformation of a matrix.

complete region of allowed scheduling vectors for the three projection vectors in combination with the two types of edges of the local DG by using Eq. 48 and Eq. 52.

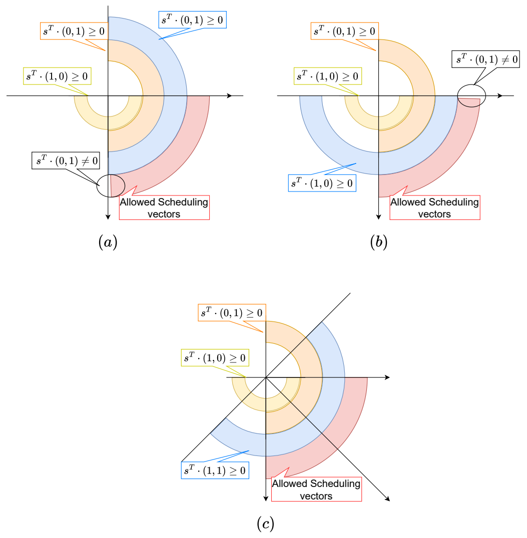


FIGURE 26: The complete derivation of the scheduling vectors restriction for the three projection vectors in Fig. 25. (a) is when **d** = (1,0), (b) **d** = (0,1) and (c) **d** = (1,1).

Once the scheduling vector $\mathbf{s}^T$ is defined, Eq 42 adds the scheduling information of the data in the DG to the SFG. Additionally, the number of the unit delays (D^n) on the SFG edges must also be considered. The number of unit delays is defined as,

$$n = \mathbf{s} \cdot (\mathbf{v}_{\text{dest.}} - \mathbf{v}_{\text{source}}) \tag{53}$$

For example, taking the projection vector **d** equal to (1,0) from the local DG in Fig. 23, and the scheduling vector $\mathbf{s}^T$ equal to the projection vector **d**, all the edges pointing downward ($\mathbf{e}_{\text{d}}$ = (1,0)) mapped to the SFG will consist of one delay unit in the SFG, except for the set of input edges. The time instance to consume and produce data through the edges of the resulting SFG is computed by Eq. 42. Fig. 27 illustrates the fully annotated SFG with the scheduling information of the data and the delay units on the edges for the projection vector **d** and scheduling vector **s** set to (1,0).

APPENDIX E TESTBENCH AND TEST FLOWCHART

A. TESTBENCH

The testbench consists of two components, e.i. the Device-Under-Verification (DUV) and the Test-Vector-Controller (TVC). The correlator array with all its internal components is realized in VHDL and instantiated as DUV. The testbench and the TVC are also described in VHDL. The functional behavior verification and the gate-level timing simulation of the DUV are conducted in Questasim. Firstly, the DUV functional behavior is verified before being synthesized by Quartus Prime Standard. During the simulation, the generated input vectors are imported by the TVC component

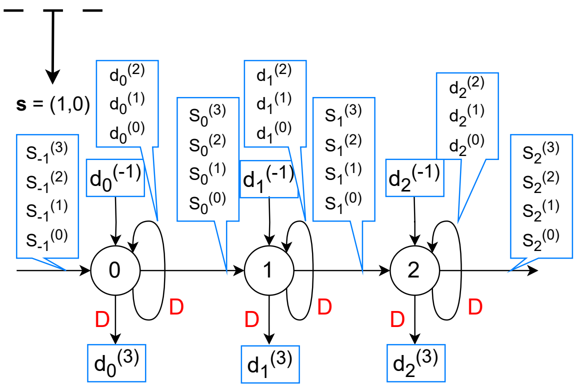

s = (1,0)
$S_{-1}^{(3)}$ $S_{-1}^{(2)}$ $S_{-1}^{(1)}$ $S_{-1}^{(0)}$
$d_0^{(2)}$ $d_0^{(1)}$ $d_0^{(0)}$
$d_0^{(-1)}$
$S_0^{(3)}$ $S_0^{(2)}$ $S_0^{(1)}$ $S_0^{(0)}$
$d_1^{(2)}$ $d_1^{(1)}$ $d_1^{(0)}$
$d_1^{(-1)}$
$S_1^{(3)}$ $S_1^{(2)}$ $S_1^{(1)}$ $S_1^{(0)}$
$d_2^{(2)}$ $d_2^{(1)}$ $d_2^{(0)}$
$d_2^{(-1)}$
$S_2^{(3)}$ $S_2^{(2)}$ $S_2^{(1)}$ $S_2^{(0)}$
0
1
2
D
$d_0^{(3)}$
$d_1^{(3)}$
$d_2^{(3)}$

FIGURE 27: The fully annotated Signal Flow Graph.

and fed into the inputs of the DUV. Comparatively, the TVC exports the output vectors of the DUV into text files. The vectors of these exported text files are compared to the output vectors of the behavioral model in MATLAB for verification as illustrated in Fig. 28. Moreover, the TVC component also provides the clock, the reset, and the enable signals for the registers in the DUV during the simulation. After the last data of an input vector is processed in the DUV, the TVC component is set to terminate the simulation. Thus, the product of the clock period with the length of the output vectors determines the simulation run-time on Questasim. After the pre-synthesis DUV functional behavior is verified, the correlator array architecture is synthesized, and the produced gate-level netlist is instantiated as the DUV. The standard delay file and the tcl script (*dump all vcd nodes*) are loaded on the DUV additionally in QuestaSim for the gate-level timing simulation. The execution of the gate-level timing simulation is conducted similarly to the simulation of the pre-synthesis design. The generated input vectors used for the pre-synthesis design are fed to the DUV with the gate-level netlist incorporated. The resultant output vectors in this case are also verified with the output vectors of the behavioral model in MATLAB. After the gate-level timing simulation, the *Value Change Dump* file is produced by QuestaSim. This file is fed to the Power analyzer tool of Quartus Prime Standard and the power report is produced.

The pseudo-code of the correlator behavior MATLAB model in Fig 28 is generalized in Alg. 2, where the output vectors *real_real*, *imag_imag*, *real_imag* and *imag_real* are exported. The symbol (⊗) is denoted as the behavioral model of an 8-bit recursive multiplier using 16 elementary 2×2 building blocks written on MATLAB.

B. THE TEST FLOWCHART.

A shell script is provided to run the test setup automatically for collecting the hardware resources, power, and the maximum operating frequency. The basic flowchart of the shell script is illustrated in Fig. 29. The shell script only works in a Linux environment with Intel Quar-

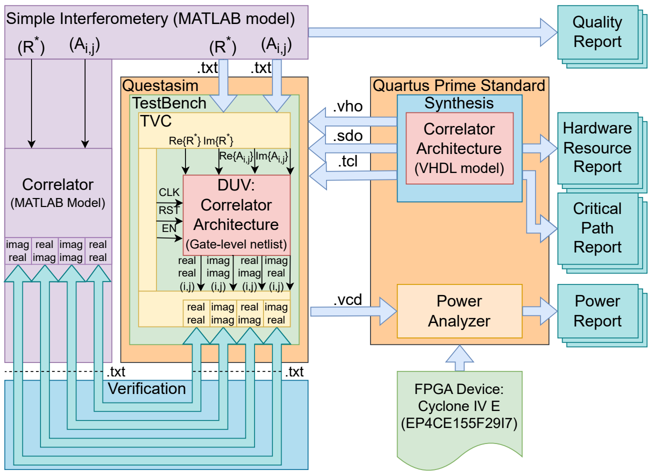

Simple Interferometery (MATLAB model)
Quality Report
Correlator (MATLAB Model)
Questasim
TestBench
TVC
DUV: Correlator Architecture (Gate-level netlist)
CLK
RST
EN
Quartus Prime Standard
Synthesis
Correlator Architecture (VHDL model)
.vho
.sdo
.tcl
.vcd
.txt
Hardware Resource Report
Critical Path Report
Power Analyzer
Power Report
Verification
FPGA Device: Cyclone IV E (EP4CE155F29I7)

FIGURE 28: The test setup with the testbench embedded for collecting the hardware resource and power result.

Algorithm 2 Pseudo-code of the simple Interferometry (MATLAB model) in Fig. 28.

```
1: exportInputVector(real(conj(R)), imag(conj(R))
2: for $i = 1 : N$ do
3:     for $j = 1 : N$ do
4:         exportInputVector(real(A{i,j}), imag(A{i,j}))
5:         for $t = 1 : T$ do
6:             real_real$\{t\}$ += real$(A\{t,i,j\}) \otimes$ real(conj$(R\{t\})$)
7:
8:             imag_imag$\{t\}$ += imag$(A\{t,i,j\}) \otimes$ imag(conj$(R\{t\})$)
9:
10:            real_imag$\{t\}$ += real$(A\{t,i,j\}) \otimes$ imag(conj$(R\{t\})$)
11:
12:            imag_real$\{t\}$ += imag$(A\{t,i,j\}) \otimes$ real(conj$(R\{t\})$)
13:        end for
14:        exportOutputVectors(real_real, imag_imag, real_imag, imag_real, i, j)
15:    end for
16: end for
```

tus Prime Standard and Questasim installed. For gate-level timing simulation, Questasim must be configured with Altera's and Cycloneive's netlist libraries. Both libraries can be obtained from Intel's ModelSim software. The shell script, the VHDL files, and the netlist libraries are provided at https://gitlab.utwente.nl/s2210703/correlator-architecture.git.

APPENDIX F OPERATING FREQUENCY IN FPGAS

In any digital circuit, there exist many paths connected with a number of logic gate components from an input source to an output destination. The path with the most logic gate components is known as the critical path. The data travel time on the critical path indicates how fast the circuit can operate. Hence, the maximum operating frequency (F_{max}) is the reciprocal of this data travel time. Synthesis tools generally provide the possibility to measure the data travel time on all paths of the design for an FPGA device when Flip-Flops (FFs) are placed between all I/O pins. In any digital design, the following constraint must hold to avoid timing violation [38],

$$T_{\text{FF-to-FF}} \leq T_{deadline} \tag{54}$$

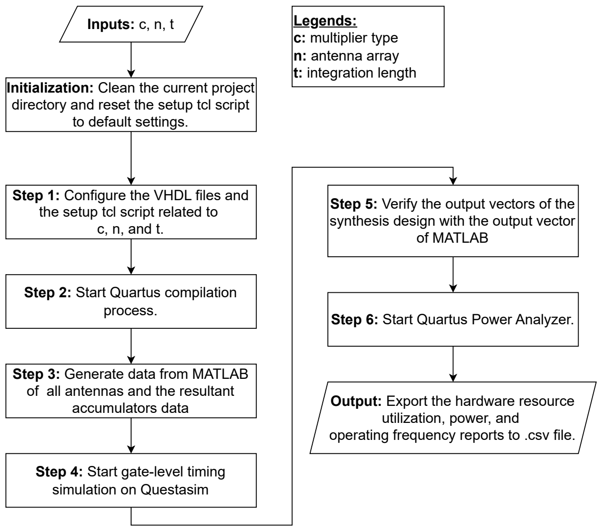

Inputs: c, n, t
Legends:
c: multiplier type
n: antenna array
t: integration length
Initialization: Clean the current project directory and reset the setup tcl script to default settings.
Step 1: Configure the VHDL files and the setup tcl script related to c, n, and t.
Step 2: Start Quartus compilation process.
Step 3: Generate data from MATLAB of all antennas and the resultant accumulators data
Step 4: Start gate-level timing simulation on Questasim
Step 5: Verify the output vectors of the synthesis design with the output vector of MATLAB
Step 6: Start Quartus Power Analyzer.
Output: Export the hardware resource utilization, power, and operating frequency reports to .csv file.

FIGURE 29: The test flowchart

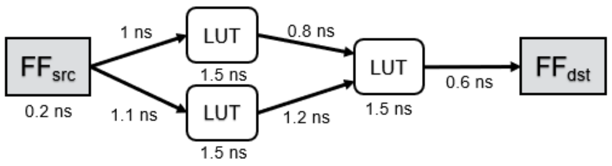


FIGURE 30: Different paths between the same FFs [38]

$T_{\text{FF-to-FF}}$ is the time delay between the source FF and the destination FF. T_{deadline} is the deadline that the data must arrive at the destination FF. For timing analysis, the FPGA tool-chain refers to only two components on a path between a source FF and a destination FF. These two components are called cells and interconnect cells. A cell is a name representation of an FPGA resource, such as FFs, LUTs, CLB/LAB, DSP, embedded memory, etc. An interconnect cell is a name representation of a routing element, such as the routing wires, the connection boxes, and the switch boxes. The delay between a source FF and a destination FF is a summation of all cell delays (T_C) and interconnect delays (T_{IC}), that form a path between them. Note that every FF has a small delay to output (Q) the stored value after a clock edge ($T_{\text{clk-to-Q}}$). Considering FF as a cell, $T_{\text{clk-to-Q}}$ is essentially the T_C of FF. The total time delay on a path between FFs is derived as

$$T_{\text{FF-to-FF}} = T_C + T_{IC} \tag{55}$$

For example, Fig. 30 shows multiple paths between the same source FF and destination FF of a design. T_C of both paths equals to 3.2 ns whereas the T_{IC} of the longest path is 2.9 ns. This will result in $T_{\text{FF-to-FF}} = 3.2ns + 2.9ns = 6.1ns$ and an operating frequency of 164 MHz.

The deadline is influenced by three factors. The first factor is the clock period of the clock signal on the FPGA (T_{clk}).

This clock period is decided during the system design, but it can vary to find the maximum operating frequency. The second factor is the setup time (T_{setup}) of the destination FF. This is the time, that the data must be stable before the rising/falling edge of the clock signal to avoid producing a metastable output from the FF. The third factor is the time skew of the clock signal arriving at the source FF and destination FF (T_{skew}). Note that T_{setup} minimizes the deadline by requiring the data to be stable before a clock edge while the T_{skew} increases the deadline due to the small delay between the FFs, hence

$$T_{deadline} = T_{clk} + T_{skew} - T_{setup} \tag{56}$$

The difference between T_{deadline} and $T_{\text{FF-to-FF}}$ is referred as the setup slack (S_{setup}) where the following statements are derived:

1) Setup violation exist in the design when $S_{setup} < 0$;
2) The F_{max} is determined when $S_{setup} = 0$;
3) All paths meet the setup time for all FFs when $S_{setup} \geq 0$.

The naming convention of Intel Quartus Timing Analyzer [39] compared to Eq. 54 is that the Data Arrival Time is equivalent to the sum of $T_{\text{FF-to-FF}}$ and the Source Clock Delay, the Data Required Time to T_{deadline} and the T_{skew} to the Destination Clock Delay subtracted by the Source Clock Delay.

A. THE CRITICAL PATH OF THE CORRELATOR ARRAY ARCHITECTURE ON THE FPGA

It is assumed the reader knows the correlator array of N×N architecture from Section IV and V. From the derived dependency graph in Section IV, it can be observed that the critical path contains one multiplier and T number of adders. The latency (L) of a m-bit input recursive multiplier structure is estimated as [27],

$$L_{m\times m} \approx max(L_{\frac{m}{2}\times\frac{m}{2}}) + L_{adder,m\times m} \tag{57}$$

Thus, the internal critical path of the 8-bit input recursive multipliers consists of one elementary 2-bit input multiplier block ($L_{2\times 2}$), the adders in the 4-bit input multiplier block ($L_{adder,4\times 4}$), and the adders in the 8-bit multiplier ($L_{adder,8\times 8}$). The critical path in the "*addi*" component consists of a comparator ($\leq$), a multiplexer (MUX), and a ripple-carry-adder (RCA) as illustrated in Section V.

To measure the operating frequency of the circuit, it is required to place registers on the input and output (I/O) on all paths by Quartus Prime Timing Analyzer. The input registers must be driven by the same clock as the output registers to compute the maximum operating frequency (F_{max}) by Quartus Prime Timing Analyzer. Note that the reported F_{max} is measured from the clock input port and ignores the user-specified clock periods. For the measurement of finding the critical path and latency, the clock period set on the clock ports is 25 ns (40 MHz).

Table 9 illustrates the critical path of the correlator array on the FPGA from Quartus Prime Timing Analyzer with the number of cells, and interconnect cells (ICs), that each component on the critical path contributes. The source FF is the input register and the destination FF is the accumulator. The 8-bit input recursive multipliers (RM) types are from Table 1. As can be observed, the internal critical path in the 8-bit input multipliers has the same amount of cells and ICs for all 8-bit RM structures in Table 1. The difference in the number of cells and ICs usage occurs at the "*Addi*" component. The ISH1 has one Cell and one IC on the MUX component, and 20 on the RCA. The ISH12 has one Cell and one IC on the MUX component, and 21 on the RCA.

In Table 10, the timing information is collected from the critical paths in Table 9 using the Quartus Prime Timing Analyzer tool. The speedup (%) is taking the reported F_{max} of the correlator array architecture with an accurate RM structure against the F_{max} of correlator array architecture embedded with the approximate RM as,

$$\text{Speedup} = (1 - \frac{\text{Accurate RM}_{F_{max}}}{\text{Approximate RM}_{F_{max}}}) \cdot 100\% \tag{58}$$

It can be observed, that the critical path of the correlator array architecture with the ISH1 embedded consists of the least amount of cells and ICs, but still has a lower slack compared to the correlator architecture embedded with the accurate RM and negative speedup of 1.13 %. Moreover, it also gives a negative speedup on the correlator array architecture. It can be observed that the correlator array architecture with the FPGA_ISH4 embedded does gain a positive speedup and has the same number of cells and ICs on the critical path as the accurate RM. Therefore, the latency of using approximate RM on the correlator array architecture is not a dominant component in speedup when realized on Intel's Cyclone Clone IV E FPGA. When designing an approximate multiplier for a correlator architecture on Intel's Cyclone Clone IV E FPGA, the synthesis-tool method of mapping the circuit design from the VHDL to the FPGA must be considered. This way the benefits of applying approximate computing on Intel's FPGAs, regarding speedup, can be exploited.

From the dependency graph derived in Section IV, we define that the critical path contains one multiplier and T number of adders. To investigate this on the FPGA, Table 11 and 13 are presented.

Table 11 illustrates the critical path of correlator array architecture of 1×1 embedded with the accurate RM structure for the set of integration lengths (T) = {2^1,2^2,2^3,2^4, 2^5,2^6,2^8,2^{10},2^{12}}. Table 13 illustrates the critical path of the correlator array architecture of 1×1, 2×2, 4×4, 6×6, 8×8, 10×10 embedded with the accurate RM structure and integration length set to 64.

From Table 11, we can observe the 8-bit input multiplier component contributes to the same amount of cells and ICs for all the integration lengths in {2^1,2^2,2^3,2^4, 2^5,2^6,2^8,2^{10},2^{12}} As T increases, the number of cells and ICs increases by the RCA component. When the integration length is set to T^1, the number of cells and ICs follows the output bit-width (W-bit) in Section V, which is 17-bit. It is expected that as the integration length increases in 2^z, an addition of z-bit is added to the W-bit. Hence, a z number of cells and ICs is expected to be added to the critical path by the RCA component. However, this is not the case for the rest integration lengths. Table 12 provides the timing information on the critical paths in Table 11. From Table 12, the maximum operating frequency indeed decreases as the integration length increases.

In Table 13, the critical paths consist of the 8-bit input accurate RM and the circuits in "*Addi*", namely the comparator, the multiplexer, and the ripple-carry adder. The number of cells and ICs on the critical path contributed by the 8-bit multiplier is the same amount for the correlator array of 1×1, 2×2, 4×4, 6×6, 8×8, 10×10. The number of cells and ICs contributed by the RCA component on the critical path varies for the correlator array size of 2×2, 8×8, and 10×10. The lowest number of cells and ICs contributed by the RCA component occurs when the correlator array size is equal to 2×2. Table 14 provides the timing information on the critical paths in Table 13. It can be observed from Table 14, that as the correlator array size increases, $T_{\text{FF-to-FF}}$ increases, the setup slack and the F_{max} decreases. A negative slack is reached on the critical path when the correlator array size is 10×10. This is due to $T_{\text{FF-to-FF}}$ being longer than the clock period assigned on the clock signal, where the solution is to change the clock period to a lower $T_{\text{FF-to-FF}}$. Even though the critical path consists of similar components for the correlator array of 1×1, 2×2, 4×4, 6×6, 8×8, 10×10, the decrease of F_{max} comes from the cells and ICs duration (T_C and T_{IC}) on the FPGA. According to Amagaski M. and Shibata Y. [37], the amount of traffic through the external routing track on the FPGA is reduced when the number of logic stages is small. Essentially, T_{IC} on the critical path grows on a larger design since the amount of traffic through the external routing track increases. Because of the increase in traffic in the routing track, the synthesis tool maps a longer physical route on the FPGA between cells to avoid congestion. In Table 14, T_{IC} indeed increases as the correlator array size increases, thus a longer route on the critical path between cells is mapped by the synthesis tool.

TABLE 10: Extra timing information on the critical path in Table 9.

Multiplier Type	**Total Cells**	**T$_C$** (ns)	**Total ICs**	**T$_{IC}$** (ns)	**T$_{FF\text{-}to\text{-}FF}$** (ns)	**Slack** (ns)	**Fmax** (MHz)	**Speedup** (%)
Accurate	40	11.474	39	10.6	22.306	3.019	45.49	0
Conv1	40	11.618	39	10.793	22.643	2.719	44.88	-1.36
Conv2	40	11.113	39	11.493	22.838	2.515	44.47	-2.29
ISH1	39	11.902	38	10.479	22.613	2.768	44.98	-1.13
ISH2	40	11.149	39	10.935	22.316	2.621	44.68	-1.81
FPGA_ISH1	40	12.004	39	10.554	22.79	2.118	43.7	-4.10
FPGA_ISH2	40	11.482	39	10.798	22.512	2.777	45.0	-1.09
FPGA_ISH3	40	11.12	39	11.578	22.93	2.391	44.45	-2.34
FPGA_ISH4	40	11.616	39	10.363	22.484	2.849	45.51	0.04
FPGA_ISH5	40	11.602	39	11.336	23.2	1.744	43.0	-5.79

TABLE 9: The critical path from Quartus Prime Timing Analyzer tool for the correlator array of 1×1 and integration length of 64.

Multiplier	**FF**	$L_{2\times2}$	$L_{2\times2}$	$L_{\text{Adder},4\times4}$	$L_{\text{Adder},4\times4}$	$L_{\text{Adder},8\times8}$	$L_{\text{Adder},8\times8}$	$(\leq)$	$(\leq)$	**Mux**	**Mux**	**RCA**	**RCA**	**FF**	**FF**
Type	Cell	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**
Accurate	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
Conv1	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
Conv2	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
ISH1	1	1	1	6	6	6	6	3	3	1	1	20	20	1	1
ISH2	1	1	1	6	6	6	6	3	3	1	1	21	21	1	1
FPGA_ISH1	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
FPGA_ISH2	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
FPGA_ISH3	1	1	1	6	6	6	6	3	3	1	1	21	21	1	1
FPGA_ISH4	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
FPGA_ISH5	1	1	1	6	6	6	6	3	3	1	1	21	21	1	1

TABLE 11: The critical path of the correlator array of 1×1 with an 8-bit input accurate recursive multiplier for a given integration length (T).

Multiplier	**FF**	$L_{2\times2}$	$L_{2\times2}$	$L_{\text{Adder},4\times4}$	$L_{\text{Adder},4\times4}$	$L_{\text{Adder},8\times8}$	$L_{\text{Adder},8\times8}$	$(\leq)$	$(\leq)$	**Mux**	**Mux**	**RCA**	**RCA**	**FF**	**FF**
Type	Cell	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**
Accurate (T=2^1)	1	1	1	6	6	6	6	3	3	0	0	17	17	1	1
Accurate (T=2^2)	1	1	1	6	6	6	6	3	3	0	0	17	17	1	1
Accurate (T=2^3)	1	1	1	6	6	6	6	3	3	0	0	18	18	1	1
Accurate (T=2^4)	1	1	1	6	6	6	6	3	3	0	0	20	20	1	1
Accurate (T=2^5)	1	1	1	6	6	6	6	3	3	0	0	20	20	1	1
Accurate (T=2^6)	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
Accurate (T=2^8)	1	1	1	6	6	6	6	3	3	0	0	24	24	1	1
Accurate (T=2^{10})	1	1	1	6	6	6	6	4	4	1	1	23	23	1	1
Accurate (T=2^{12})	1	1	1	6	6	6	6	2	2	1	1	27	27	1	1

TABLE 12: Timing analysis of the correlator array of 1×1 with an 8-bit input accurate recursive multiplier for a given integration length (T) from Table 11

	Total Cells	**T$_C$** (ns)	**Total ICs**	**T$_{IC}$** (ns)	**T$_{\text{FF-to-FF}}$** (ns)	**Slack** (ns)	**Fmax** (MHz)
Accurate (T=2^1)	35	11.026	34	10.593	21.851	3.123	45.71
Accurate (T=2^2)	35	11.466	34	10.247	21.945	3.003	45.46
Accurate (T=2^3)	36	11.674	35	10.944	22.850	2.075	43.62
Accurate (T=2^4)	38	11.216	37	10.635	22.083	2.843	45.13
Accurate (T=2^5)	38	11.222	37	10.434	21.888	3.047	45.55
Accurate (T=2^6)	40	11.474	39	10.6	22.306	3.019	45.49
Accurate (T=2^8)	40	11.756	39	10.880	22.868	2.467	44.38
Accurate (T=2^{10})	43	12.038	42	12.077	24.347	0.992	41.65
Accurate (T=2^{12})	44	11.973	43	12.366	24.571	0.764	41.26

TABLE 14: Timing analysis of the correlator array of N×N with an 8-bit input accurate recursive multiplier and an integration length of 64 from Table 13

	Total Cells	T_C (ns)	**Total ICs**	T_{IC} (ns)	$T_{\textbf{FF-to-FF}}$ (ns)	**Slack** (ns)	**Fmax** (MHz)
Accurate (N=1)	40	11.474	39	10.6	22.306	3.019	45.49
Accurate (N=2)	38	11.908	37	10.731	22.871	2.503	44.45
Accurate (N=4)	40	11.725	39	11.668	23.625	1.706	42.93
Accurate (N=6)	40	11.427	39	12.797	24.456	0.831	41.38
Accurate (N=8)	39	10.738	38	14.110	25.08	0.237	40.38
Accurate (N=10)	40	11.358	39	14.769	26.359	-1.084	38.34

TABLE 13: The critical paths of the correlator array of N×N with an 8-bit input accurate recursive multiplier and an integration length of 64.

Multiplier	**FF**	$L_{2\times2}$	$L_{2\times2}$	$L_{\text{Adder},4\times4}$	$L_{\text{Adder},4\times4}$	$L_{\text{Adder},8\times8}$	$L_{\text{Adder},8\times8}$	$(\leq)$	$(\leq)$	**Mux**	**Mux**	**RCA**	**RCA**	**FF**	**FF**
Type	Cell	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**	**Cell**	**IC**
Accurate (N=1)	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
Accurate (N=2)	1	1	1	6	6	6	6	3	3	0	0	20	20	1	1
Accurate (N=4)	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
Accurate (N=6)	1	1	1	6	6	6	6	3	3	0	0	22	22	1	1
Accurate (N=8)	1	1	1	6	6	6	6	3	3	0	0	21	21	1	1
Accurate (N=10)	1	1	1	6	6	6	6	3	3	1	1	21	21	1	1