Implementation of DOA Estimation Algorithm Based on FPGA

Hengyuan Zhou[1], Xiaojun Jing[1(✉)], Bingyang Li[2], Zesheng Zhou[3], and Bogan Li[4]

[1] Beijing University of Posts and Telecommunications, Beijing, China
{zhouhengyuan,jxiaojun}@bupt.edu.cn
[2] Jiangsu Automation Research Institute, Lianyungang, Jiangsu, China
[3] Shandong Institute of Aerospace Electronics Technology, Yantai, Shandong, China
[4] University of Southampton, Southampton, UK
Bohan.Li@soton.ac.uk

Abstract. Direction of arrival (DOA) estimation is the prerequisite for beamforming, which largely determines the performance of smart antennas. However, DOA estimation algorithms usually need a large amount of computation while making enormous demands on real-time processing, which poses challenges to hardware implementation. Field Programmable Gate Array (FPGA) is widely used in the implementation of DOA estimation algorithms in recent years due to its advantages of high throughput rate, parallelizable computing, and design flexibility. But the traditional method of using hardware description language (HDL) to implement algorithms on FPGA has disadvantages such as high complexity and long time-to-market. Since the 21st century, high-level synthesis (HLS) tools have gradually developed, allowing designers to define their algorithms with a higher abstraction level so that effectively reduces workload and development time.

Keywords: Direction of arrival · MUSIC · FPGA · HLS

1 Introduction

The direction of arrival (DOA) estimation is of great significance and is used in many applications. DOA estimation can enhance the sensing ability of the communication system, such as sensing the direction of the vehicle relative to the roadside unit (RSU) in the vehicular communication [1] and sensing the location of the device in the communication system in which the unmanned aerial vehicle participates [2, 3]. But DOA estimation generally requires a large amount of computation. Researchers initially used the digital signal processor (DSP) chips to implement the algorithm. The estimated time is at the level of milliseconds [4, 5]. Recently, with the development of field programmable gate array (FPGA), the advantage of parallel computing has emerged. Special designs such as confidentiality can also be realized. More importantly, the delay jitter of algorithms on FPGA is weak, which is very suitable for implementing various communication algorithms. By designing flexible, fast, stable, and parallel processing algorithms, FPGAs have gradually replaced DSP chips in the field of DOA estimation, compressing DOA estimation time to the microsecond level [6].

S. Sun et al. (Eds.): ICSINC 2021, LNEE 895, pp. 46–52, 2022.
https://doi.org/10.1007/978-981-19-4775-9_7

The shortcomings of hardware description language (HDL) have gradually been exposed with the geometric growth of the circuit scale. First, these languages are verbose and error-prone, have rough syntax. What's more, they may produce sub-optimal, faulty hardware, which is usually difficult to debug [7]. High-level synthesis (HLS) tools enable engineers to use software programs to specify hardware functions. Compared with HDL, the abstraction level raises from the register-transfer level (RTL) to the algorithm or behavior level. NEC's research shows that it takes about 300,000 lines of HDL code to realize a million-gate FPGA design, and the use of modern HLS tools can easily increase the code density by 7 to 10 times, requiring only 30,000 to 40,000 lines of code [8]. HLS tools enable algorithm engineers to use high-level languages to develop FPGA algorithms without fully grasping relevant hardware knowledge. This paper verifies the feasibility of using C language to implement the MUSIC algorithm in Vivado HLS and explores the effect of parallel optimization instructions.

2 Design of HLS Project

2.1 Algorithm Implement

MUSIC algorithm parameters implemented on FPGA: the number of antenna elements is 4, the number of signals is 2, the number of snapshots is 128, and the resolution is 1° so that there are 181 space spectrum points. The variables and main loops are shown in Table 1 and Table 2 respectively. The specific process is as follows (Fig. 1):

Table 1. Variables used in the implementation process

Name[Size]	Data
a_theta[4][181]	Direction vector
X[4][128]	Received signal
Rx[4][4]	The covariance matrix of X
RxU[4][4]	The feature vector of Rx
En[4][2]	Noise subspace
a[4][1]	Direction vector in a certain direction
aH_En[1][2]	Product of the conjugate transpose of the direction vector and the noise subspace
aH_En_EnH_a[1][1]	Square of the aH_En's norm
P_theta[181]	Space spectrum
log_P_theta[181]	The logarithm of the space spectrum

Table 2. The main loops in implementation

Name	Function
calculate_P_theta	Calculate space spectrum
read_a_theta	Read direction vector in a certain direction
calculate_log_P_theta	Calculate the logarithm of the space spectrum

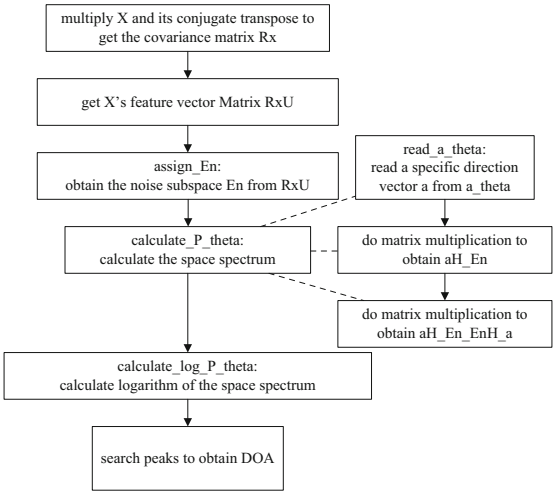

Fig. 1. Algorithm implementation flowchart

1. Call the matrix multiplication function to multiply the received signal matrix (X) and its conjugate transpose to obtain the covariance matrix (Rx).
2. Perform eigenvalue decomposition on the covariance matrix (Rx). Get the eigenvector matrix (RxU) of the covariance matrix.
3. Take the last 2 columns of the eigenvector matrix to get the noise subspace matrix (En).
4. Calculate the space spectrum (calculate_P_theta). The inner loop obtains the direction vector array (a) through the loop named read_a_theta. Then the outer loop multiplies the conjugate transpose matrix of direction vector and En to calculate array aH_En. Multiply aH_En and its conjugate transpose to calculate aH_En_EnH_a, and store aH_En_EnH_a in the space spectrum array (P_theta).
5. Search peaks on the space spectrum array to obtain DOA.

All matrix multiplications in the algorithm are implemented by the matrix multiplication function named matrix_multiply provided in Vivado HLS, which can easily realize various matrix transposition and parallel optimization. Eigenvalue decomposition is the most important and most complex part of the MUSIC algorithm. In this experiment, we call the svd_top function in the Vivado HLS algorithm library using the bilateral Jacobi

method to realize the eigenvalue decomposition. The bilateral Jacobi method is more accurate than other singular value decomposition methods and is convenient for parallel calculation, so it is more suitable for implementation on FPGA.

2.2 Parallel Optimization

The optimizations directives for loops in Vivado HLS mainly include pipelining, unrolling, merging, and dataflow. Pipelining adds registers before devices used in the loop so that a cycle can start after the previous one releases the necessary resources. Several adjacent cycles overlap in time, greatly improving resource utilization and reducing the delay. Unrolling allocates several times the hardware resources required for a single cycle to the entire loop so that multiple cycles can be performed on different hardware resources at the same time to reduce the time consumption.

The loop to calculate space spectrum (calculate_P_theta) is unrolled by 2 times and pipelined. The inner loop reading direction vector (read_a_theta) is fully unrolled. The loop to calculate the logarithm of the space spectrum is pipelined.

Partition is the most significant directive for arrays. Due to the access bottleneck of a single block of RAM, a single array is divided into several pieces and stored in different blocks of RAM or registers in a specific way, which can improve the throughput of the array. Partition methods include block partition, cyclic partition, and complete partition, etc.

The direction vector array (a) used to calculate the space spectrum needs to be read and written frequently, therefore it is completely partitioned and stored in the registers to speed up the operation. The matrix multiplication operation of the direction vector (a) and the noise subspace matrix (En) can be decomposed into the multiplications of array a and each column of En. The operation can be accelerated by reading an entire column of the noise subspace matrix. Therefore, the noise subspace matrix (En) is divided so that the elements in the same row are stored in the same block of RAM.

3 Simulation and Analysis

3.1 Accuracy Compared with MATLAB

The simulation uses a 4-element uniform linear array with a half wavelength element spacing. Its number of snapshots is 128. The SNR is 20 dB. Space spectrums generated by FPGA simulation and MATLAB simulation are shown in Fig. 2. They almost coincide. The difference mainly appears where the value is small. When FPGA performs floating-point numbers or some non-linear calculations, accuracy may be lost. But it does not affect the DOA estimate result.

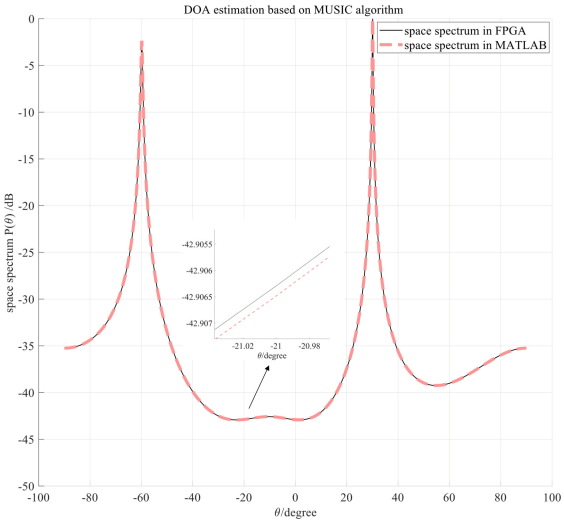

Fig. 2. Space spectrums generated by FPGA simulation and MATLAB simulation

3.2 Estimation Speed

The difference before and after optimization is mainly manifested in the number of resources and clock cycles required by each loop and function.

Figure 3 shows the delays of the main loops before and after the optimization. The division of the noise subspace matrix and the unrolling in the matrix multiplication function cause Vivado HLS to automatically divide the noise subspace matrix completely and store it in the registers. Compared with the default solution, the single-cycle delay of the loop (calculate_P_theta) has been reduced from 41 clock cycles to 22 clock cycles. The initial interval (II) before and after optimization is 41 clock cycles and 4 clock cycles respectively, and the number of cycles drops from 181 to 90. Ideally, the initial interval should be 1 clock cycle when the inner loop is fully unrolled. But if it is fully unrolled, it will take up too many resources and not meet the resource constraints. Partial unrolling, retaining part of the serial operation in the matrix multiplication makes II reduce to 4, and the delay of the entire loop body is reduced from 7421 to 379.

The loop to calculate the logarithm of the space spectrum (calculate_log_P_theta) involves non-linear floating-point operations such as division and logarithm. If optimization is not performed, the delay will be huge. After optimization, the iteration latency remains unchanged. The II is reduced from 28 to 1 clock cycle, and the number of cycles is reduced from 181 to 90. Therefore, the overall delay of the loop body is reduced from 5068 to 117, and the delay is reduced by more than 97%.

As shown in Fig. 4, in comparison to the result before optimization, although the usage of program block memory, look-up tables, registers, and DSP blocks all increased, the overall latency dropped from 29023 to 16501, a drop of more than 40%.

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- assign_En_outer	24	24	6	-	-	4	no
+ assign_En_inner	4	4	2	-	-	2	no
- calculate_P_theta	7421	7421	41	-	-	181	no
+ read_a_theta	8	8	2	-	-	4	no
+ a_col_loop	12	12	6	-	-	2	no
- tranverse_log_P_theta	360	360	2	-	-	180	no
- calculate_log_P_theta	5068	5068	28	-	-	181	no

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- assign_En_outer	32	32	8	-	-	4	no
+ assign_En_inner	6	6	3	-	-	2	no
- calculate_P_theta	379	379	22	4	1	90	yes
- tranverse_log_P_theta	540	540	3	-	-	180	no
- calculate_log_P_theta	117	117	28	1	1	90	yes

Fig. 3. Delay of the main loops before and after optimization

Utilization Estimates

	s_default	s_optimized
BRAM_18K	22	23
DSP48E	690	889
FF	55049	69027
LUT	56978	70398
URAM	0	0

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
Verilog	Pass	29023	29023	29023	NA	NA	NA
Verilog	Pass	16501	16501	16501	NA	NA	NA

Fig. 4. Resources utilization and entire latency before and after optimization

4 Conclusion

In the process of implementing the MUSIC algorithm, the advantages of HLS tools can be summarized in the following four points. First, it uses a high-level language instead of HDL, which has a higher level of abstraction and a smaller workload. Second, HLS tools are similar to an algorithm development environment, rather than a hardware development environment. Thirdly, the HLS tool uses independent optimization instructions to control FPGA synthesis, which allows developers to get different synthesis results without modifying the source code. Finally, there are abundant algorithm library resources in HLS tools. They can be easily called and integrated into various designs. In summary, using HLS tools to write FPGA prototypes of DOA algorithms such as MUSIC is very attractive. It can be verified relatively quickly and obtain considerable performance, which is an effective method.

References

1. Mu, J., Gong, Y., Zhang, F., Cui, Y., Zheng, F., Jing, X.: Integrated sensing and communication-enabled predictive beamforming with deep learning in vehicular networks. IEEE Commun. Lett. **25**, 3301–3304 (2021)
2. Gao, N., Li, X., Jin, S., Matthaiou, M.: 3-D deployment of UAV swarm for massive MIMO communications. IEEE J. Sel. Areas Commun. **39**(10), 3022–3034 (2021)
3. Gao, N., Jin, S., Li, X., Matthaiou, M.: Aerial RIS-assisted high altitude platform communications. IEEE Wirel. Commun. Lett. **10**(10), 2096–2100 (2021)
4. Huang, Q.: The research of DOA estimation algorithm based on DSP. University of Electronic Science and Technology of China (2010)

5. Zhou, Z.Y.: The research and implementation of multi-source super-resolution DOA estimation algorithm based on DSP system. Huazhong University of Science and Technology (2012)
6. Liu, T.: Implementation of classical DOA algorithm based on FPGA. Harbin Institute of Technology (2016)
7. Zwagerman, M.D.: High-level synthesis, a use case comparison with hardware description language. Grand Valley State University (2015)
8. Wakabayashi, K.: C-based behavioral synthesis and verification. ASP-DAC (2004)