Rapid development of SAR backward projection imaging algorithm based on MATLAB

Jiaqi Zhang¹, Lianghai Li and Zhenhua Zhang
Beijing Research Institute of Telemetry, Beijing 100076, China

¹ E-mail: jiaqi_zhang1993@163.com

Abstract. This paper introduces a kind of FPGA rapid development method to complete the backward projection algorithm of SAR. Firstly, the traditional FPGA development has strict requirements for developers' software and hardware knowledge. Then we introduce a kind of FPGA rapid development method based on the 2018(a) version of MATLAB's Simulink of MATLAB. And then, the algorithm is divided into two parts, one is the range pulse compression part, and the other is the backward projection part. In order to make full use of the parallelism of BP algorithm, this paper divides the dates into four parts in the distance direction and the imaging grid is also divided into four parts. The projection grid is still based on the whole grid, which can improve the real-time performance of the algorithm. After the algorithm is modeled in Simulink, the model can be converted to the corresponding FPGA code by using the fixed point tool. The comparison of Scope display results with MATLAB results demonstrates the effectiveness of this MATLAB-based Simulink tool for FPGA development.

1. Introduction
The traditional SAR imaging mechanism is implemented by FPGA+DSP. The DSP has good computing performance, but its parallelism is poor, and it is difficult to achieve real-time imaging. FPGA has good parallelism and flexibility, but its development is relatively cumbersome. Therefore, it is difficult to implement SAR imaging algorithm with FPGA by using traditional FPGA programming. And the traditional FPGA development design method firstly requires the designer to realize floating-point algorithm simulation in MATLAB and convert it to fixed-point operation. Then, the designers need to write the FPGA development language by themselves, and finally the function verification is repeated to verify the correctness of the code. This requires the software designer to have a good knowledge of the hardware and hardware engineers have a good knowledge of the algorithm, which requires the designer to master both the high-level abstract MATLAB language and the low-level FPGA language. Moreover, these two levels of conversion will not only take a lot of time, but also the correctness of the two stages [1-3].

Figure 1. FPGA development process based on Simulink.
Simulink is the toolbox of MATLAB, its main function is building system models, simulation and analysis. We can simulate and analyze the models before the system is built completely, and to adjust the system parameters in real time [4]. This can improve the performance and stability of the system and improve the development efficiency in the system design process. The FPGA development process based on Simulink in this paper is shown in Figure 1. First, the corresponding Simulink model is built according to the simulation algorithm in MATLAB, and then the model is fixed, and finally, the model automatically generates the FPGA code.

This development method based on Simulink's direct conversion to FPGA code makes it possible to implement SAR imaging algorithms on FPGAs. To a certain extent, it provides a new idea for the development of FPGA.

2. Algorithm design method

The backward projection algorithm (BPA) designed in this paper is shown in Figure 2 [5]. The BPA is completed into two parts. The first part is the process of range pulse compression of the original echo data, and the second part is the process of backward projection. The second part is shown in Figure 3: first we should calculate the corresponding distance of the current projected grid to each pulse(①), and find the coherence coefficient value corresponding to the distance(②), and then according to the distance we can get the two way delay into the address to find the pulse compression result(③). Then we need to take the result and the coherence coefficient corresponding to do multiple and accumulate(④). The coherent accumulation process of the projection grid to each pulse is completed until all the imaging grids are traversed. These two processes need to be performed sequentially. The first process uses the traditional frequency domain pulse compression mode. The process of range pulse compression has been realized in reference 7. It should be noted that the results of the pulse compression should be stored on-chip for being repeated called later. The second process is written in the form of a state machine in Figure 3.

According to the simulation design, a and b in Figure 4 represent grid point coordinates, in which ais the azimuth direction, and b is the distance direction, and the maximum values are amax and bmax. And m represents the nth pulse echo with a maximum value of mmax. The imaging mesh size of this algorithm is 1024*256 pixels, so the range of a is 1:256, the range of b and 1:1024, and the range of m is 1:256. The output RESULT represents the imaging result of the pixel. According to the principle and steps of the BPA, the general idea of the Veriolog implementation module based on Simulink to construct the BPA is as follows.

state 0: In the initial state, we have known that the sampling clock is clk, and each calculation is completed as the rising edge of the clock. In the initial state, when a=1 and b=1, the imaging processing is at the first pixel of the projection grid, it jumps to state 1.

state 1: First, we should calculate the distance between the pixel (a, b) and the pulse m, and then calculate the two-way delay and the value of the coherent scattering coefficient which is displayed as exp in Figure 4 according to the distance, and then calculate the address of the corresponding pulse according to the delay, and take out the value which is the echo data according to the address, that is result in Figure 4. Next, the exp is multiplied by the result to obtain the result, and the result is superimposed on SUM. In state 1, the values of a and b remain unchanged, and the value of m is added from 1 to 256, and when m = 256, it indicates that the imaging result of the current pixel has been obtained, and jump to state 2.

state 2: At this time, SUM has superimposed all the 256 values of the current pixel point (a, b), that is, the imaging result of the current point has been obtained, and the value of SUM is assigned to the RES, and the imaging of the pixel point (a, b) is completed. At this time, the values of a and b should be judged to determine whether the pixel has been traversed to the entire grid. If a=256, b=1024, it indicates that the last pixel imaging has been completed, and jumps to state 3; If b=1024 and a<256, it indicates that the imaging of the a column has been completed, and next we need to calculate the a+1 column. So we take a=a+1, b+1, and jump to state 1; If b<1024 and a<256, it means that the
calculation of the pixel of the a column has not been completed. So we take \( b = b + 1 \), the value of \( a \) remains unchanged, and jump to state 1.

state 3: The BPA pixel processing has been completed by traversing the entire projected pixel point.

As is shown in the reference [6], the BPA has good parallelism, so this paper divides the distance data 1024 into 4 sub-blocks by distance partitioning, and each sub-block contains 256 distance-echo data. So the data size of the sub-block is 256*256 (R*A). The reasons for blocking in the distance direction are that, firstly since each pixel needs to coherently accumulate the total number of pulses in the azimuth, it is not suitable for the BPA to perform block processing in the azimuth direction. And then since the target slant distance is much larger than the azimuth moving distance, the pixel points and the echoes in the distance direction correspond to each other, and thus the data can be divided in the distance direction. The processing of the echoes in the four sub-blocks is the same. And the difference is that the input and distance of the signal are offset from the centre of the imaging grid of each sub-block relative to the centre of the original imaging grid. Therefore, the processing of the four sub-blocks can generate the same FPGA code, and can be executed in parallel when processed by the FPGA, which can improve the real-time efficiency.

![Figure 2. The flow chart of BPA.](image)

![Figure 3. The flow chart of backward projection.](image)
Figure 4. Overall design block diagram of the Imaging calculation module for BPA.

3. Simulink modules of BPA

3.1. Data source generation module
The original echo of the Simulink-based BPA used in this paper has been down-converted and has been quantized in 16-bits. The echo data can be gained from MATLAB, which constitutes a 1024*256 matrix `sig` with a total of 262144 dates. According to the development process of 2.1, firstly, the whole data is subjected to range pulse compression. Then we need to convert `sig` into a one-dimensional vector of one size of 262144, which can be done in MATLAB by: `B2 = reshape ( sig, 1, 1024*256)`.

3.2. Range pulse compression module
The "PC" module built in this paper mainly include FFT module "FFT", complex multiplication module "comp_mult" and IFFT module "IFFT". The "FFT" and "IFFT" can be implemented in the same structure. The Simulink module of the pulse compression module designed in this paper is shown in Figure 5. Where `B2` is the result of the data source generation module and `res_PC` is the pulse compression result.

Figure 5. The detail of "PC" module.

Since the result `res_PC2` will be called repeatedly in the addressing module, this article stores it in the on-chip dual-port RAM with a RAM depth of 262144. Therefore, the written address is 0~262143, this requires a considerable amount of FPGA storage resources. In the find addressing module, the address of the pixel is calculated as the read address of the RAM, and then find the result of the value of the echo scattering response.
Reference [7] has pointed out that the "FFT" Module of the Xilinx FPGA IP core has four structures: pipeline, Streaming I/O structure, base-4, Burst I/O structure, base-2, Burst I/O structure, base-2, Simplified Burst I/O structure. The pipelined Streaming I/O structure has good data throughput and is suitable for large data volume. Therefore, the pipeline structure is used to implement the FFT function.

3.3. Imaging calculation module
In the imaging processing of BPA, the echo after "PC" is divided into four sub-blocks, each sub-block contains 256 distance echoes, and the azimuth is full aperture. The corresponding projection mesh is also divided into four sub-grids, which is equivalent to each sub-grid corresponding to a part of the original mesh, and the entire imaging calculation is still performed with respect to the original mesh. The imaging calculation module of the distance-to-block BPA designed in this paper is shown in Figure 6. The internal structures of the four imaging sub-modules are identical. The first sub-module of the BPA will be analyzed mainly below with the detailed block diagram of the sub-module.

It can be seen from the enlarged Figure 6 that each sub-module contains 5 input ports and 1 output port, where the input ports input_a_1, input_a_2, input_a_3, input_a_4 are 4 echo signals generated by the "PC" module. They will be used as input sources for the four "imaging calculation modules": BP_sub_1, BP_sub_2, BP_sub_3, BP_sub_4; The input port A indicates the distance to the grid of the current sub-module imaging grid and the entire imaging grid; And the input ports wr_addr indicates the write address of the RAM, and wr_en indicates the write enable of the RAM, and the en_BP indicates the enable of the "Imaging calculation" module. The three input ports are the same for the four sub-modules, and they are controlled by the counter. As shown in Figure 7. The output port result represents the result of the current sub-module Subsystem.

Figure 6. The "Imaging calculation" module of BPA.

Figure 7. The "Subsystem1" module of the four sub-module in Figure 5.

The counter HDL min Figure 7 is the write address of the RAM, and its value is from 0 to 262144. The value of the counter HDL m1 determines whether read or write of the RAM: when it is between 0 and 262143, the write enable wr_addr is valid, and the value of the input input_a_1 is written
according to the current value of the address \( wr \_addr \); when it is greater than 262143, the \( en \_BP \) is valid, at the same time, the "find_addr" module starts to calculate the RAM read address. The \( BP \_sub \_1 \) in Figure 6 is shown in Figure 8.

**Figure 8.** The details of "Subsystem" module.

In Figure 8, the "find_addr" module represents the addressing calculation module, and its input is the enable \( en \_BP \) and the grid difference \( A \). And its output \( addr \) is the read address of the RAM module, and its another output \( R \_ij \) represents the distance of the obtained imaging pixel to the Azimuth Phase Centre (APC), as is shown in Figure 9. The Simple Dual Port RAM is the RAM that stores the distance-pulse-compressed datas. We need to write or read the datas according to the access rules described above; The "exp" module represents the calculation of the coherence coefficient module based on the delay \( R \_ij \) as is shown in Figure 10, we need to multiply the distance \( R \_ij \) by the phase factor to obtain the coherence coefficient. After we get the echo datas and the coherence coefficient we can get the final output result by the multiplication module "Multiply_Accumulate".

**Figure 9.** The details of "find_addr" module in Figure 7.
The principle of the distance $R_{ij}$ in "find_addr" module is that: $R_{ij}=\sqrt{((R(b))^2+(A(a)-\text{tan}(m)*v)^2)}$. Where $R(b)$ represents the distance position of the pixel point $(a,b)$. $A(a)$ represents the azimuth position corresponding to the pixel point $(a,b)$, and $\text{tan}(m) * v$ represents the position of the pulse $m$. Then the address can be gained by: $\text{addr}=256*(m-1)+(t-T\text{start})/\text{delta}_R-1$. The address found by the above manner is the address of the current pixel, so it needs to be manipulated as follows: $\text{addr}=256*(m-1)+\text{addr}$. This can convert it into a vector between 0 and 262143.

![Figure 10](image10.png)

**Figure 10.** The details of "exp" module in Figure 7.

4. Simulation results and code generation

In order to see the results of the Simulink model clearly, this paper uses a model with a point target at the centre of the scene. The echo contains a total of 1024*256 (Range * Azimuth) dates, and the grid point size is set to 1024*256 (Row * Column), and $a_{\text{max}}=256$, $b_{\text{max}}=1024$, $m_{\text{max}}=256$. Since the point target is set at (513, 127), the result should be at the centre of the grid. Figure 11 shows the display result of the scope. It can be seen that the BPA can generate a target echo at the centre of the grids, that is, the model simulation result is correct.

At the same time, in order to facilitate the analysis of the simulation results, the results will be displayed in the scope and workspace respectively and the result is displayed in Figure 12(a). The BPA imaging results obtained by MATLAB simulation are shown in Figure 12(b). Comparing the Figure 12(a) with 12(b), we can see that the accuracy of Simulink modeling and MATLAB simulation is similar, but the datas are different. This is because the decimal part is rounded when Simulink simulation is performed. And in order to save resources, the data type has been converted from double to fixed points, so that the values are approximated.

![Figure 11](image11.png)

**Figure 11.** The result of BPA displayed in scope.

![Figure 12](image12.png)

**Figure 12.** The result of BPA in Simulink (a) and in MATLAB(b).
Then we use the HDL Coder tool to automatically convert the code into FPGA. The code selected in this article is Verilog code. The generated HDL project is shown in Figure 13(a), and the converted code is shown in Figure 13(b). The Verilog code corresponding to any module can be found in the Code Generation Report. This is convenient for reading and calling, and is easy to modify and maintain.

![Figure 13](image)

**Figure 13.** The result of HDL project (a) and HDL Code (b).

5. Conclusions
This paper introduces the MATLAB-based tool Simulink to complete the implementation process of SAR imaging algorithm on FPGA. The result proves the effectiveness of using high-level language for automatic transcoding of low-level languages. This not only solves the cumbersome development process of the existing FPGA and the long development cycle, but also improves the real-time performance of the algorithm and lays a foundation for future FPGA development. And in this way, when there is a problem, you can directly look up the model timing from the model, no need to look up from the bottom of the code, making the development from the bottom to the model-based top-level code development.

References
[1] Yang W M, Liu Y L and Liu L H 2011 *Journal of Tianjin University of Science & Technology*. **26**(6) 65-68
[2] Hua L F, Wan S B and Wei X S 2010 *Information Technology* (10) 41-43
[3] Fan Q X 2015 The research on advanced development technology of FPGA (Beijing: Beijing Institute of Technology University)
[4] Liu D Y 2014 The Real-time imaging processing design of SAR based on Z7.(Xian: XIDIAN University)
[5] Desai M D, Jenkins W K 1992 *IEEE Transactions on Image Processing* **1**(4) 505-517
[6] Lu Y N 2014 Design of parallel computing System for back projection algorithm (Nanjing: Nanjing University)
[7] Li A, Yu Y, Chu C and Zhang Z H 2017 *Journal of Telemetry, Tracking and Command* **38**(2) 13-21