# Design, Implementation and Performance Evaluation of Synthetic Aperture Radar Signal Processing on FPGA

by

Hemang Parekh

B.E. (Electronics) Maharaja Sayajirao University of Baroda, Vadodara, India, 1998

Submitted to the Department of Electrical Engineering and Computer Science and to the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science `in Computer Engineering.

_____

Professor in Charge

_____

_____

Committee Members (2)

_____

Date of Acceptance

Dedicated to

my Mother and my Late Father

## Acknowledgements

I would like to offer my respect and sincere appreciation to my advisor Dr. Gary Minden who has helped me throughout my graduate studies at KU and now with my thesis. Specifically I would like to thank him for being patient and keeping his faith in me during last summer when I struggled overcoming the grief of sudden loss of my Father. I would like to thank Dr. Joe Evans for all the support and help during the course of the project and for the classes that I had opportunity to take under him. My special thanks to Dr. Arvin Agah who has obliged to be on my committee.

I would like to thank Ed Komp for all the help and support that he provided during the course of the Adaptive Computing Systems project. I would like to thank him specifically for helping me in the debugging process even during the weekends.

I vividly recall all those late evenings spent in ACS labs with my other colleagues Sarin, Sandeep, Karthik and Pramod discussing ideas to overcome various software and hardware problems we faced during the course of this project. I would like to thank all of them for such wonderful time in ACS project.

Above all, it is because of my mother's blessing and my sister's love, that I have made it to this stage; I do not have words to express my gratitude.

## Abstract

Synthetic Aperture Radar Image Generation from raw data is an application, which demands heavy computational power and large memory for operation. Researchers are exploring various hardware solutions to improve the performance and speed over existing software solutions. This thesis specifically deals with such image generation algorithm being implemented by Alaska SAR Facility (ASF) through software in C. This work offers an FPGA based prototype solution using FLASH language and Block Diagram Editor. Both these tools are developed as a part of the ACS project at the University of Kansas, sponsored by DARPA. This thesis also demonstrates the ease and flexibility of design using FLASH and BDE. The result shows that this current implementation is a good prototype design demonstrating the feasibility of an FPGA based SAR signal processor.

## CONTENTS

## List Of Figures

## List Of Tables

# 1 Introduction

## *1.1 Motivation*

Synthetic Aperture Radar has been used extensively for various military, commercial as well as scientific purposes since it is instrumental in providing target detection and tracking, terrain imaging, ocean currents, sea ice motion, weather mapping, vegetation analysis, etc. It is obvious that such remote sensors designed for global coverage will generate an immensely large amount of data. Over and above that, to generate interpretable images from these data would require extensive signal processing. 15s of data from ERS-1 amount to some hundreds of MBs of data and to digitally process this large amount of data in real-time would require a computer to do computations of the order of GFLOPs. Presently various software implementations for processing stored data are available. [Pryde94] analyses the system requirements for a real-time signal processor for such SAR data.

Custom hardware solutions could offer the required real-time performance but it would not be a solution for such small production volumes and moreover it is not flexible enough for research applications. Field Programmable Gate Arrays (FPGAs) offers the much-needed flexibility while keeping the economics behind the research significantly low. Hence this work aims at demonstrating a reasonably flexible implementation of such SAR signal processor using the Functional Programming Environment (FPE) developed at the University of Kansas as part of the ACS project funded by DARPA[1]. This thesis, also a part of this initiative, aims at demonstrating the wide range of capabilities the FPE offers in terms of flexibility and ease of design.

Just as there are a variety of Synthetic Aperture Radars in the skies, by ESA, NASA, etc, there are also various ways to perform Synthetic Aperture Radar Signal Processing. The

---

[1] DARPA contract number DABT63-97-C-0032 as part of the Adaptive Computing Systems initiative.

algorithm used here is one that is being used at the Alaska SAR Facility, in their software implementation of the processor for ERS satellite. It is discussed in detail in Chapter 2. Throughout this report, the distinction between the *algorithm & software implementation* and the *hardware implementation* should be kept in mind. The implementation that will be described was developed at University of Kansas as part of the ACS project. It is a method used to realize the algorithm in hardware using the Functional Design Environment through FLASH and Block Description Editor (BDE).

### *1.2 Thesis Layout*

This remaining part of this chapter discusses the merits of FPGA implementation for such memory intensive and computationally complex algorithms and looks into other such works being carried out by various researchers. Chapter 2 describes the concepts of Synthetic Aperture Radar Signal Processing. Chapter 3 discusses about the Target Architecture and the design flow adopted for this work. Chapter 4 elaborates on the FPGA implementation of FFT while Chapter 5 discusses design and implementation issues and discusses the various space-time utilization details of the implementation. Results are covered in Chapter 6, while in the concluding chapter some limitations to the algorithm are mentioned with possible explanations. The thesis ends with some suggestions for future work. The Host program for controlling the FPGA working and various BDE diagrams are available in the Project Technical Report [ACS_Report2000]. Also this report includes the various BDE diagrams for generating bit files for SAR application.

### *1.3 Field Programmable Gate Arrays*

FPGAs are arrays of gate, offering the elementary resources of storage, logic and wires in a standard part, which can be programmed at the time of use, to make an application specific circuit. Depending on the vendors the actual FPGA chip level architecture differs, but conceptually they are an array of static RAM which holds the controls, logic gates, storage elements to define a functional unit. FPGAs offers some great benefits like instantaneous implementation (since loading a design on FPGA is with a second),

dynamic reconfiguration (reprogramming a part of the FPGA at run-time), design security (on power down, FPGA configuration is lost) and field programmability.
In case of Xilinx FPGAs, the smallest logic unit is Constraint Logic Block (CLB). By gluing such logic units together, larger applications can be structured and restructured on this same piece of silicon. This reconfigurability makes FPGAs a better solution over custom-made integrated circuits.

### *1.4 Related Works*

There are some efforts made by researchers to model SAR processors on hardware, which uses FPGAs.

The RASSP architecture developed at Advanced Technologies Laboratories, Lockheed Martin [Pridgen95, Zuerndorfer94, RASSP_SAR] was used for implementing a SAR signal processor for an airborne SAR. FPGAs were used for control signals and as a hot rod FPGA directly interfaced to HOT ROD fiber optic board, which gets the SAR video signal directly. However, the image dimensions were not as large as has been used by this thesis work. It is one of the earliest efforts using FPGAs as an important processing unit for SAR signal processing.

Another effort [Muehring97] uses Mercury Computer Systems RACE multicomputers. Though this effort does not directly relate to an implementation on FPGA type of hardware resource, it is worth mentioning, because of its design of an optimal configuration for SAR processing. This effort has provided insight to us on various optimization rules and methods to be followed and mapped for FPGAs.

A more recent effort [Andraka] by Ray Andraka, on Virtex XCV1000 FPGAs on WILDSTAR boards, a product by Annapolis Micro Systems, is really a good work, as it projects the feasibility of a real time SAR processor. It implements a Doppler weather radar processor on FPGA but does not involve large data complexity equivalent to what this work tries to achieve.

# 2 Synthetic Aperture Radar Signal Processing

Radar is an active system that transmits a beam of electromagnetic (EM) radiation in the microwave region of the EM spectrum. Consequently, this extends our ability to observe properties about the surface of the earth that previously were not detectable. As an active system, the SAR provides its own illumination and is not dependent on light from the sun, thus permitting continuous day/night operation. Furthermore, neither clouds, fog, nor precipitation have a significant effect on microwaves, thus permitting all-weather imaging. Thus, we have an instrument capable of continuously observing dynamic phenomena such as ocean currents, sea ice motion, or changing patterns of vegetation. To understand the operations of SAR let us first consider a conventional radar.

## *2.1 Conventional Radar*

In traditional radar, sending a pulse of constant frequency with $\tau_p$ duration will offer a range resolution $\delta_R$ given by,

$$d_R = \frac{ct_p}{2} \qquad \Lambda \ (2.1)$$

where *c* is speed of light.

This shows that for improving range resolution, $\delta_R$, $\tau_p$ must be as small as possible. However to maintain a significant level of SNR in the received signal, a high total power in the transmitted signal becomes essential. Hence transmitting smaller $\tau_p$ and large total power would result into high burs of energy, which is impractical for most systems. AN improvement in this is possible by sending a chirp signal instead of a single frequency pulse. The range resolution in this case becomes,

$$d_R = \frac{c}{2B} \qquad \Lambda \ (2.2)$$

where B is Bandwidth of the chirp signal. Since chirp bandwidth in MHz are easily feasile, a fine range resolution is obtained.

On the other side, azimuth resolution $\delta_a$ is dependent on the real antenna aperture A, mathematically,

$$d_a = \frac{R\,l}{A} \qquad \Lambda\ (2.3)$$

where R is the range of the radar, and λ is the wavelength. Since the real antenna size is limited by the payload size and weight limits on a spacecraft, there is a significant restriction on azimuth resolution in case of a conventional radar.

### *2.2 Principles of Synthetic Aperture Radar*

As compared to a conventional radar, as first realized by Carl Wiley in 1951, in SAR, one can synthesize a much longer aperture by using Doppler spread of the echo signal. SAR provides the advantage of obtaining finer azimuth resolution even when using smaller antenna apertures, and it is even independent of the range R. Ideally its azimuth resolution $\delta_a$ is equal to half the real antenna aperture length, L.

$$d_a = \frac{L}{2} \qquad \Lambda\ (2.4)$$

Figure 2.2.1 shows the scanning geometry of a right side looking SAR antenna in a simplified format.

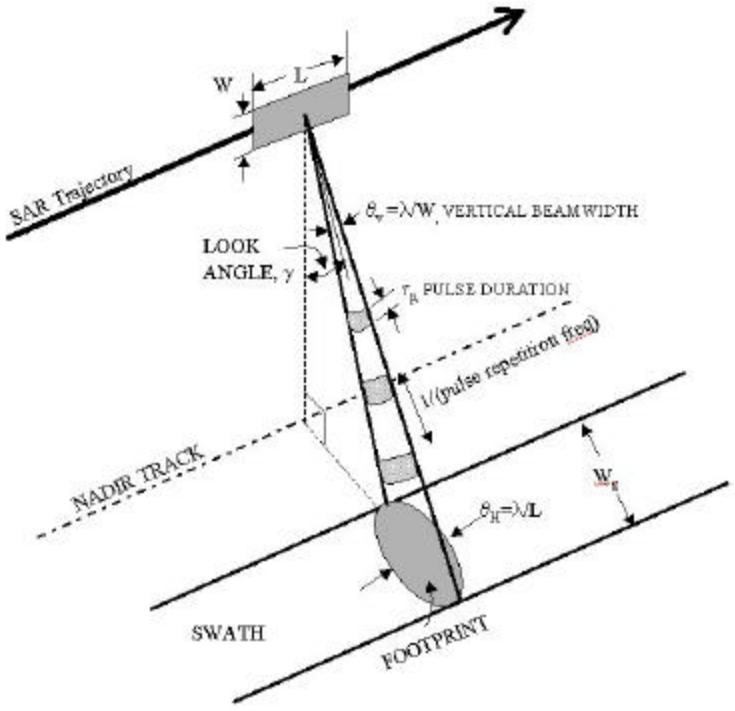


**Figure 2.2.1 Simplified SAR scan geometry (right side looking)**

However, approaching the ideal limit value of azimuth resolution is lower bounded by one fact. Since we need to measure range as well as along track position, the radar must be pulsed. When a pulse is transmitted, the radar then goes into a listening mode to detect the target echo. We then require that the time of reception of the earliest possible echo from any point in the swath, due to a particular pulse transmission be later than the time of reception of the last possible echo from any other point, due to the transmission of the previous pulse. Otherwise, we will attribute the trailing portion of the previous pulse echo to a nearby point illuminated by current pulse. This, effectively places a lower limit on the area of the antenna (A = WL).

A SAR processing system which attains its along track resolution by simple frequency filtering of the Doppler waveform is called unfocused SAR. This processor is unable to accommodate the variable rate of change of phase from a single point target. Such an unfocused SAR does provide resolution better than conventional radar, but to attain high resolution images, it is necessary to process the SAR Doppler signals in some way that can account for the variation in Doppler frequency of a target as it passes through the footprint. The result would be a focussed SAR image that approached the along-track resolution limit of L/2.

Analyzing the echo signal taking into account non-linear phase behavior, it is shown in various SAR references [Curlander91, SSUG93, ASPA, Carrara95, Levanon88, Wegmuller97], that a replica correlation or matched filtering is the solution for attaining such high resolution SAR image.

Thus, matched filtering forms the heart of SAR signal processing. Now we shall discuss the signal-processing algorithm to generate SAR image. The algorithm is the one that ASF has implementated in C. There is lots of information on their implementation on their website at http://www.asf.alaska.edu and the references [SSUG93, ASPA] gives information about their algorithm and the theory supporting it. Much of the theory explained in the next few subtopics have been abstracted from [Curlander91, SSUG93, ASPA].

## *2.3 Signal Processing Algorithm*

### Matched Filter

Although the returns from points at adjacent range intervals overlap in time, the distinctive shape of chirp pulse is sufficient for signal analysis to enable the components of the superimposed signals to be resolved. In effect, a matched filter for the emitted pulse will recognize the elements of the distinctive signal and delay them successively so that they are all compressed into a short spike with intensity proportional to that of the extended echo. Although a matched filter may seem to be the perfect solution it also needs to be considered that the resolving capability of the system is determined by the narrowness of the main lobe and its separation from, and amplitude ratio to, the sidelobes in the frequency response of the matched filter.

For ERS-1 range processing, the width is about 0.13 μsec, the separation of the first sidelobe is about 0.10 μsec, and its relative amplitude is about 0.21. Since the range sampling rate is 18.96 MHz, the sampling interval is about 0.053 μsec, which is about half the lobe separation. This implies that successive compressed pulses will have sidelobe overlap and a significant amount of the image intensity between a pixel and its neighbors will be ambiguous. To compensate for this sidelobe ambiguity, it becomes essential to weigh the returned signal over the integration time or perform windowing in the frequency domain over the reference function itself. For a fixed level of sidelobe reduction it has been shown in [Curlander91] that a Hamming window with $\alpha$=0.54 serves the purpose.

Thus, matched filtering technique would "compress" the system response to a point target back into a point. However, an ideal point target has infinite bandwidth and since radar has finite bandwidth, by linear processing of received echo signal, we can produce only a finite BW (smeared image) approximation to the observed point target.

A point target response, within interpulse period (of transmitted pulse), is dispersed by the structure of the transmitted pulse and by the multiple pulses which reach the target as the radar travels past it.

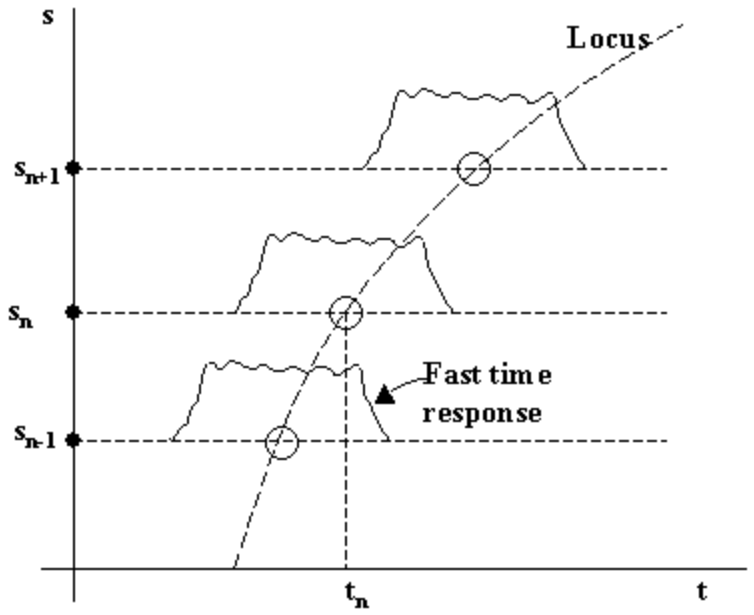


**Figure 2.3.1 2-D Point target response**

Since the range to a target could be considered constant during the time of one pulse width, a general 2 dimensional compression process, to obtain a point from the (see Figure 2.3.1) point target response, can be decoupled into two 1-D compression operation, one in fast time (response within pulse width) and other in slow time (response over period of time as satellite moves along). Since nominally, these two times map to coordinates that are orthogonal to each other, a rectangular algorithm can be applied. Range Processing to perform compression of fast time response and Azimuth Processing to perform compression of slow time response.

### Range Processing

The received signal from each pulse is correlated with the linear FM pulse replica. This procedure is repeated for each pulse for which the target was effectively in view of the radar. Consequently, the point target response becomes as shown in the Figure 2.3.2 below.

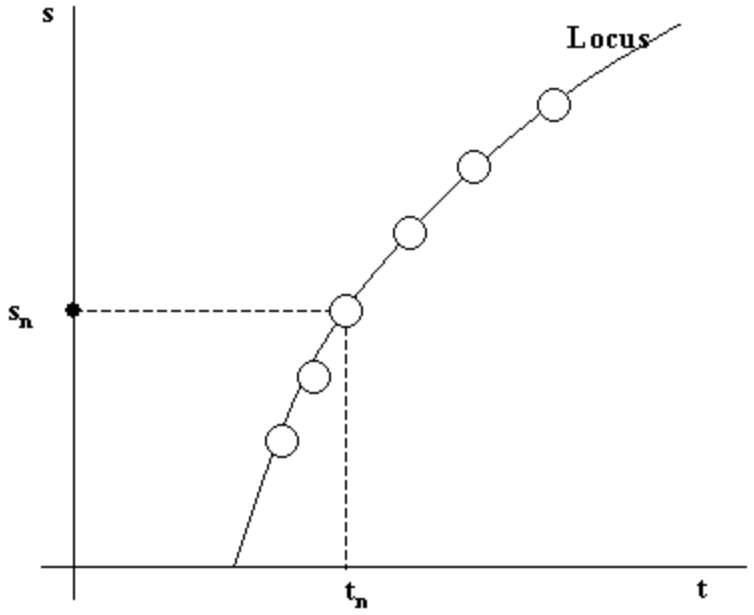


**Figure 2.3.2 Point target response after Range Processing**

Thus, the radar returns originally dispersed in 2D have now been compressed to a 1D space.

### Azimuth Processing

The signal after range compression is in fact the Doppler signal received from the point target as the radar moves by. This is called Doppler compression or azimuth compression.

The waveform is a function of various factors like orbit location, velocity of the spacecraft and most importantly it is dependent on the range of the target. Therefore, for point target at different range, we have different waveforms. So though the basic compression is of the correlator type, correlator waveform for each range needs to be calculated separately. This correlator can be implemented as a matched filter, if the parameters Doppler frequency, $f_D$ and rate of change of Doppler frequency, $f_R$ are independent of radar location over a period till the point target is visible to the radar. However, in practical cases, these change leading to range migration effect, and so a correction of range migration becomes essential.

- Range Migration

In some code signals (e.g. linear FM) Doppler shift is coupled to an additional delay. If not accounted for, inputs from a given range bin (line), can appear at the output of the matched filter, delayed, and associated with the next range bin. As long as the delay error is significantly less that pulse width after compression, the Doppler-induced range bin error is insignificant. However, in case of a SAR where the spacecraft is also moving along the observation section of its path, this delay error is inevitable. This problem is called *Range Migration.* In order to generate Doppler shift, the range shift has to be larger than the wavelength $\lambda$ and in order to remain in the same range bin, range change has to be smaller than range resolution $\Delta x$. ([Levanon88])

$$l < \Delta R < \Delta x \tag{3.4}$$

For ERS-1, the footprint being 5 km and minimum slant range, $R_0$ being 845 km the maximum difference in the slant range with respect to slant range at mid swath, $R_0$, equals to 15m. (see derivation in [SSUG93]) This is assuming a constant signal wavelength (no chirp!). This is called range curvature.

Additionally, because of the rotation of the earth, there is one more range shift, which depends on the trajectory of the satellite. This earth component adds to the swath velocity, causing added shift in the relative motion between the SAR and the target. This displacement is called range walk. This range migration path looks like a section of parabola and in fact, it is described by a quadratic equation.

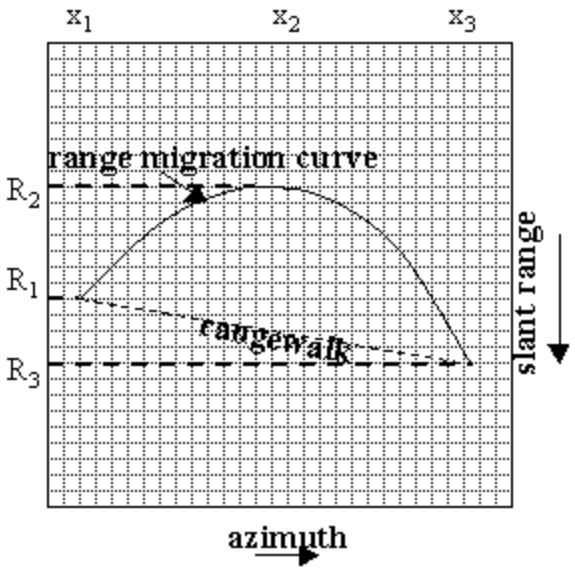


**Figure 2.3.3 Parabolic Curve traced by a point target in an azimuth-slant range plane**

The preprocessing determines the Doppler history and so the range distortions are known. This information is then used by the processor to perform range migration correction.

ASF software implementation, *aisp,* takes cares of this by providing deskewing option in the signal processing. However the current work on FPGA ignores the range walk for the sample dataset and compares the results with the software implementation by running *aisp* without setting the deskew option. The standard range-Doppler (RD) algorithm is based on the idea that all points at a particular Doppler frequency will have to be migrated by the same amount at a particular range. This is why in the RD algorithm, the migration is done after an azimuth FFT of the data is carried out. An 8-point sinc interpolation kernel is used by *aisp* to perform the interpolation.

- Azimuth Compression

After Range Migration correction, the image is now straightened out. Now the echo from a point target is scattered along the pixels along the single range bin. Thus, it seems that, an identical matched filtering as carried out in Range Compression would suffice the need. The only difference is the Doppler history along each range bin is different, so the reference function for azimuth compression will have to be different for each range bin. Since matched filtering in time domain is just a multiplication with a complex conjugate,

the azimuth reference function generated in the time domain needs to be Fourier transformed.

This azimuth reference function in the frequency domain is then weighed using an azimuth weighing function to suppress sidelobes and normalize the look energies for a multilook image. Now the azimuth scaling of the range migration corrected data is performed with the conjugate of the FFT of the azimuth reference function. The inverse Fourier transform will bring back the image to time domain.

Thus, all the above analysis shows that FFT forms a core of the algorithm, and an excellent implementation of FFT is essential to the performance of SAR processor.

As stated before Doppler Centroid estimation and Doppler Ambiguity Resolution needs to be carried out. These values are considered as available through preprocessing and directly fed to the SAR signal processor on FPGA, for this work. Some further detail about these values is provided below.

## Preprocessing

- Doppler Centroid Estimation

For the range processing, the reference function is computed from the base transmission frequency and chirp rate, while for azimuth processing, they must be determined from the Doppler shift $f_D$ and its rate of change $f_R$, together called Doppler history. Doppler Centroid is the Doppler shift at the moment the beam center crosses the target, and hence is very much dependent on the accurate measurement of the relative spacecraft to target velocity. If prior knowledge of Doppler parameters is not available, the values can be estimated from the image data itself, using *clutterlock* or *autofocus* method. ASF's *aisp* uses *clutterlock* method to estimate Doppler Centroid for the image scene. Details about these methods can be found in various references [Curlander91, Carrara95]. This FPGA implementation considers that the processing has the prior knowledge of the Doppler Centroid, $f_D$ and $f_R$ and hence further details are not provided here.

- Doppler Ambiguity Resolution

ERS-1 SAR platforms employ yaw steering of the radar in order to maintain the Doppler Centroid of the data within 1/2 of the pulse repetition frequency of the SAR. Hence the estimation of Doppler Centroid is unambiguous, Nyquist criteria being satisfied. Since the SAR processing uses discrete signal analysis, the extent of the Doppler spectrum is limited by the sampling rate, and this may lead to repetition of bright parts of the image at diminished intensity (ghosts) in the azimuth direction at intervals corresponding to multiples of the PRF.

# 3 Target Architecture and Design Flow

In this chapter, a brief description of the target hardware architecture (WILDFORCE board) and the design flow maintained during the implementation of the algorithm on FPGA is presented next.

## *3.1 WILDFORCE Board*

The WILDFORCE reconfigurable computing engine is a COTS product from Annapolis Micro Systems, Inc. It has Xilinx 4000 series of FPGAs as Processing Elements (PEs). It is a PCI-plugin board.

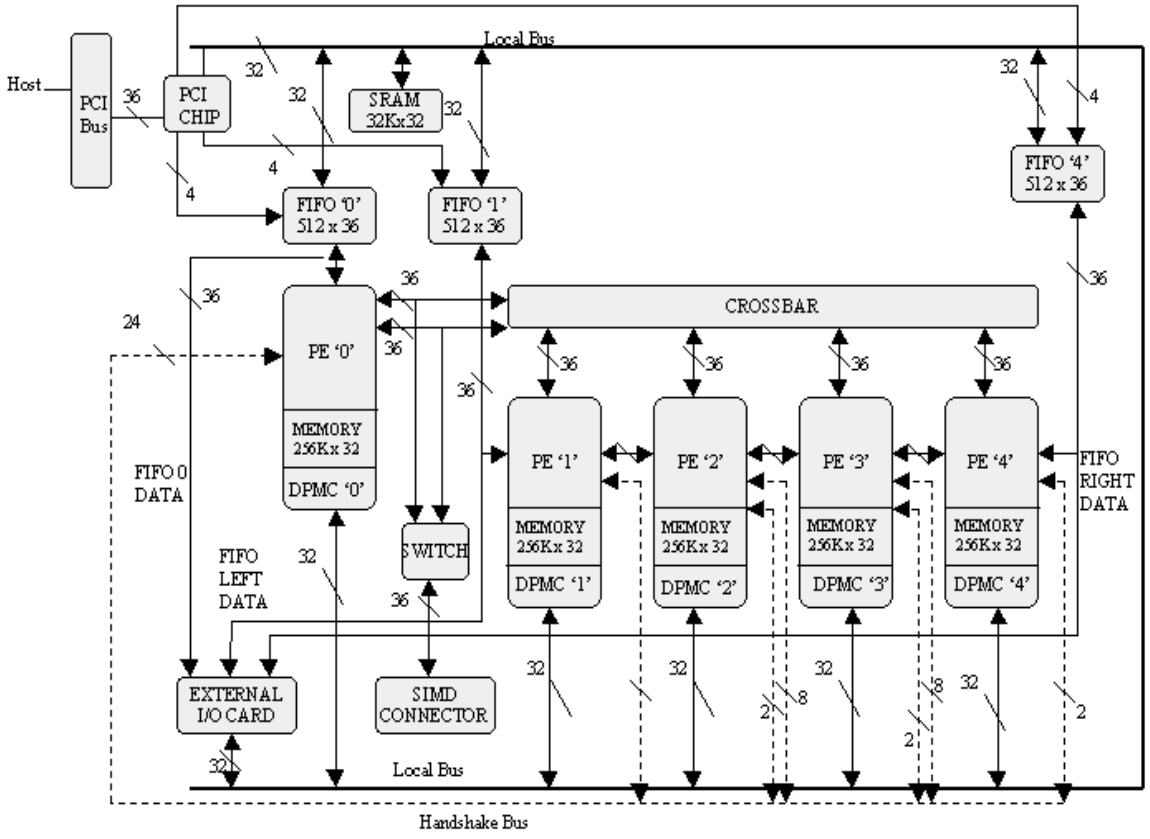


Figure 3.1.1 WILDFORCE architecture[2]

[2] Redrawn from WILDFORCE Manual

The board that we have at University of Kansas has 5 Xilinx 4085XLA FPGAs on them. Each of these PEs is equipped with a dual port 256Kx32 SRAM. These RAMs are individually accessible by the host too. The PEs can communicate with the host through the PCI bus interface, and using the interrupt API handling, various forms of data exchange can be managed. SAR application mainly uses the memory interface and interrupts are used to control the loading and reloading of the memory contents. The interface to the memory or the other mezzanine cards is implemented on the Processing Elements and the user is provided with an instantiated entity called "Logic Core" for each of the PEs.

### Memory Interface

A single WILDFORCE processing element (PE) configured with a simple memory mezzanine card uses the dual-ported memory controller (DPMC) to arbitrate accesses to memory between the host and the PE. In this way, a single ported memory may be used by both elements. Figure 3.1.2 shows a high-level block diagram of such a processing element.

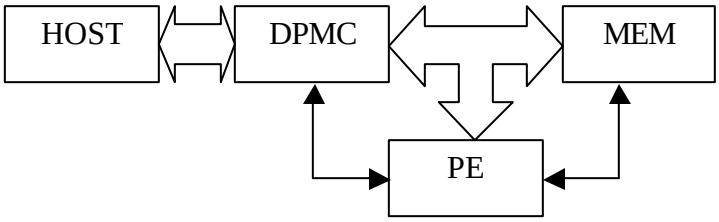


**Figure 3.1.2 Block Diagram of a single WILDFORCE PE with memory card**

- Read/Write to Memory

There are two types of APIs available for reading and writing to the WILDFORCE onboard memory. One is a regular memory read and write, and the other is for faster read/writes using DMA, by using contiguous memory locations for variables to read from and write to. This thesis compares the results with both kinds for memory access.

### *3.2 Design Flow for SAR on FPGA*

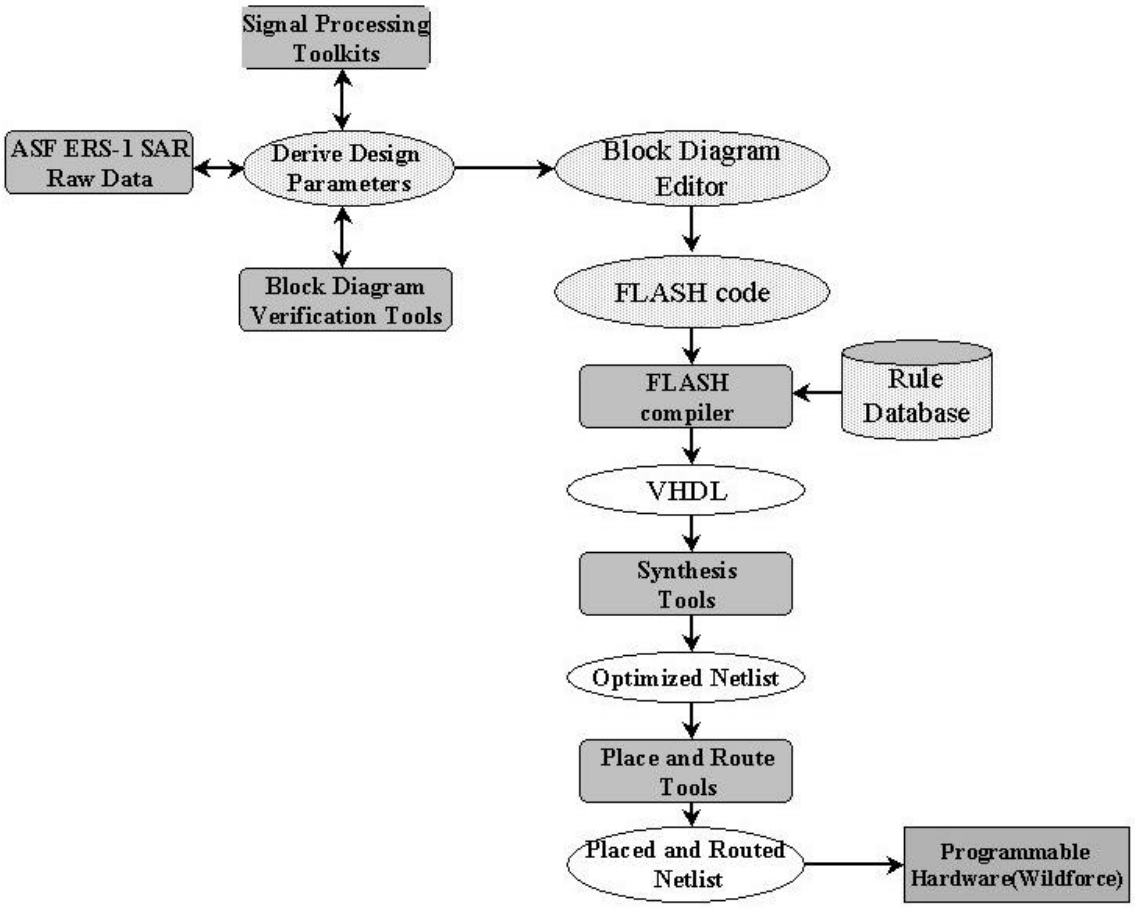


**Figure 3.2.1 Design flow for SAR on FPGA**

The design cycle starts with an analysis of the ASF software implementation of the SAR signal processor-*aisp*, simulation of matched filters in MATLAB®, study of raw data format, and parameters from metadata files and header files. At this stage, various implementation constraints are analyzed and accordingly design parameters are derived. The next stage is design partitioning. Manual partitioning is done, which is later optimized based on actual space (CLB count) and timing values from synthesized Netlist as well as place and route Netlist. The Block Diagram Editor (BDE) developed for the

ACS-FPE[3] project at the University of Kansas is used to describe the hardware implementation of the algorithms. The FLASH[4] code is generated from BDE. This code is then compiled to obtain a flattened VHDL which is then passed through conventional synthesis, place and route tools to get the final FPGA configuration bit file.

Various iterations are performed to meet space and timing constraints and correspondingly, design partitions are varied and design parameters are optimized. Figure 3.2.1 explains this flow graphically.

Some further details about BDE and FLASH is provided below.

### Block Diagram Editor (BDE)

BDE allows users to draw blocks to represent application modules, for synthesis on Xilinx FPGAs. Its basic feature is that it allows parameters to be carried to the top-level block. Parameters can be any constants to be used in the algorithm, number representation, bitwidth, etc. This allows a very general design to be made with relative ease. So one can easily design a general N point FFT, with varying bitwidths at any stage of the butterfly. Thus it allows user with the flexibility to key in parameters at the topmost level before generating FLASH and synthesizing the VHDL code generated by FLASH compiler. BDE also permits user to instantiate hand-coded VHDL as a block.

### FLASH

FLASH is a rule based language designed at University of Kansas. This higher level language allows user to generalize hardware designs. Currently FLASH compiler emits flattened VHDL. Consider the case of a multiplication. For a user to multiply two numbers one of which is a constant and the other is a signal, then FLASH compiler generates a KCM implementation of the multiplication. Moreover, the type of KCM is also dependent on the constant being used. For a constant like 7, it is economical to have

---

[3] A Functional Programming Environment of Design and Implementation of High Performance Radio and Synthetic Aperture Radar Processing Functions

[4] A Rule Based Language developed at University of Kansas for the ACS-FPE project.

multiplication with 8 ( 2nd input-signal shifted by 4) followed by subtraction of the input signal. (X*7 = X*8 -X => 1 subtractor, 1shift). A regular KCM would do 3 additions. (X*4+X*2+X). All this is based on rules which identifies the inputs to the operation, and decides which implementation is advantageous for the current case.

# 4 FPGA Implementation of Fast Fourier Transform

## *4.1 Fast Fourier Transform*

As already pointed out, SAR signal processing requires a large amount of matched filtering operation, which can be performed much faster in the frequency domain. The significant speed improvement is because of the FFT implementation of Fourier Transforms, which require only $(N/2)\log_2 N$ multiplications as compared to $N^2$ multiplications (for a radix-2 FFT). The SAR algorithm requires a continuous computation of the FFT of the range lines stored in the memory. The memory interface on WILDFORCE board is 32 bit wide. The resultant FFT Engine's interface is shown in the Figure 4.1.1 below. To maintain the generality with respect to all other modules in the SAR signal processing, only 60 lines each of 4096 values are Fourier transformed at a time. The next batch of 60 lines are requested and acknowledged by the host through the interrupt signals.

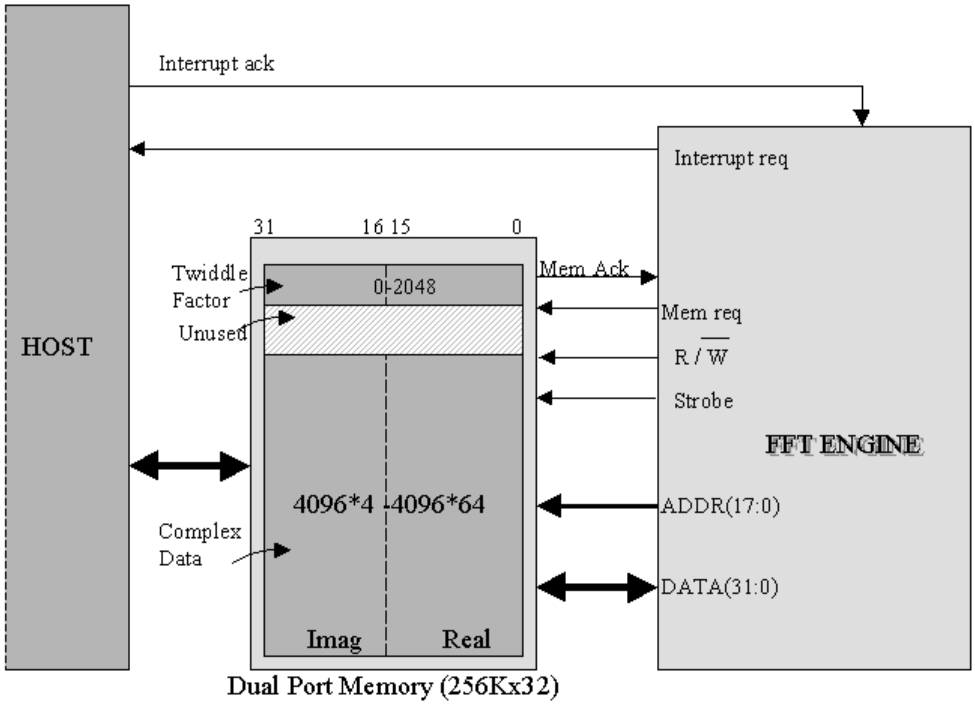


**Figure 4.1.1 Interface Diagram**

There are two basic algorithms for FFT.

1. Cooley-Tukey FFT, decimation-in-time algorithm and
2. Sande-Tukey FFT, decimation-in-frequency algorithm.

Both the methods require the same amount of computation and both can be performed as an in-place computation, using up the input memory location for output storage at each stage.

Safe-scaled FFT is also required to limit the bitwidth requirements and prevent overflows within butterflies. So before each stage of FFT, the input is scaled by 0.5.

The inverse FFT can be computed by a simple modification of the input read process. The IFFT can be obtained by computing the FFT of the following sequence:

$$X'(n) = \begin{cases} \dfrac{X(0)}{N} & for\, n = 0 \\ \dfrac{X(N-n)}{N} & for\, n = 1,2,\mathrm{K}\ ,N-1 \end{cases} \qquad (4.1)$$

Thus, the first value is stored at address 0 while the rest are stored in reverse order. This operation can easily be implemented in hardware by changing the address lines to the memory or by using an up-down counter that is properly initiated.

Another method to compute the IFFT is to first interchange the real and imaginary parts, then perform the FFT, and finally, interchange the real and imaginary parts of the output. This method is implemented for the IFFT and hence the bit file for loading the FPGA is still the same FFT bit file. Only while loading the data, the real and imaginary parts of the data values are reversed at the input and output.

### Selecting Data Word Length

The word length, with the case of SAR process in mind, was chosen to be 16 bit Imaginary and 16 bit Real. However we still need to determine the tolerable word length which we can use within the butterfly and still have good results. [Wanhammar99] discusses the noise sensitivity of safe-scaled FFT for fixed-point implementation. The variance of the output signal relates to the input signal variance according to

$$s_X^2 = \frac{1}{N} s_x^2 \quad where \quad N = FFT\ point(\ for\ this\ case\ it\ is\ 4096) \tag{4.2}$$

For large FFTs we have noise variance approximately given by,

$$s_e^2 \approx 8 s_B^2 \tag{4.3}$$

where $B$ corresponds to butterfly and $e$ for noise error.

Hence the signal-to-noise ratio is

$$\frac{s_X^2}{s_e^2} = \frac{1}{8N} \frac{s_x^2}{s_B^2} \tag{4.4}$$

Hence the SNR will be about the same at the input and output of the FFT if we use a $0.5\log_2(8N) = 7.5$ bits longer word length inside the FFT than for the input signal. We selected 25 bits for data word length

### *4.2 Implementation Details*

In place decimation in frequency Radix-2 FFT, the implementation is carried out using fixed-point arithmetic. The functional diagram is shown below along with the BDE diagram. The data format employed is 2's complement fixed point representation using 16 bits. MSB is the sign bit while the other15 bits are assumed integer bits. The WORD serial memory is used to store the 16 bit imaginary part and the 16 bit real part of the data in the upper and lower nibble of a memory location. The internal butterfly computation bus is 31 bits wide but the results are truncated to 16-bit 2's complement fixed point format. There is facility in the module to switch between scaling of the output of each stage by 0.5 to avoid overflow or to allow a normal FFT. A single butterfly takes 7 clock periods. Hence the throughput of FFT is 4096 values after 7 cycles/butterfly * $\log_2(4096)$ stages/FFT * (4096/2) butterfly/stage $\approx$ 172050 cycles approximately

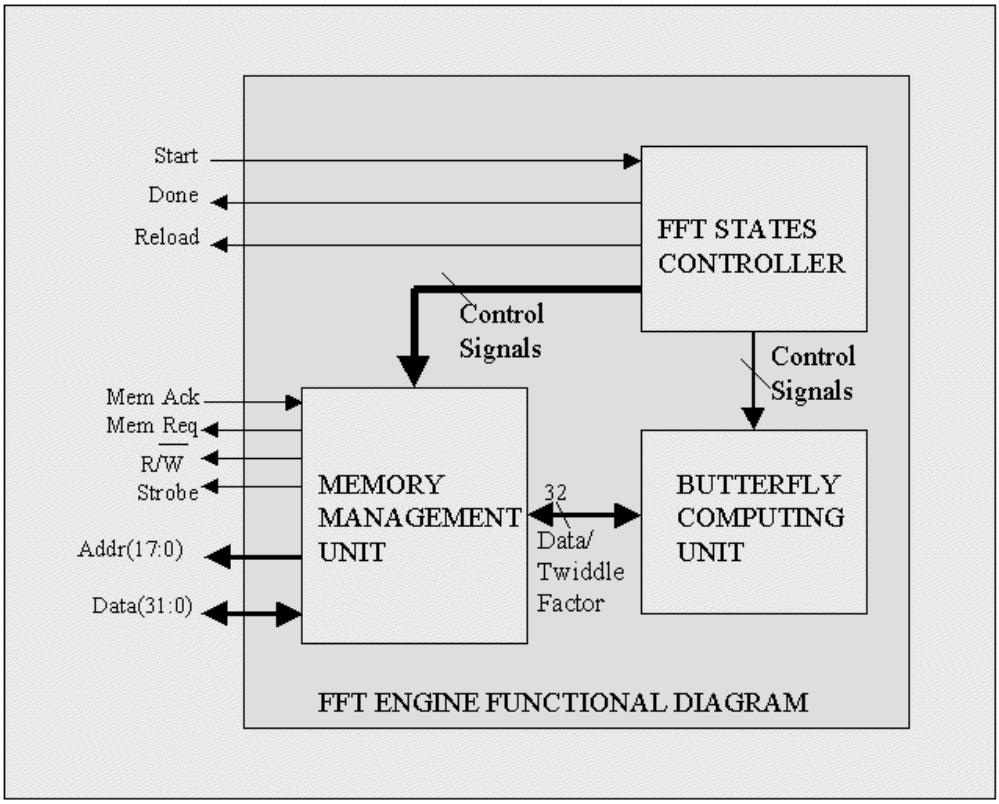


**Figure 4.2.1 Functional Diagram for FFT Engine**

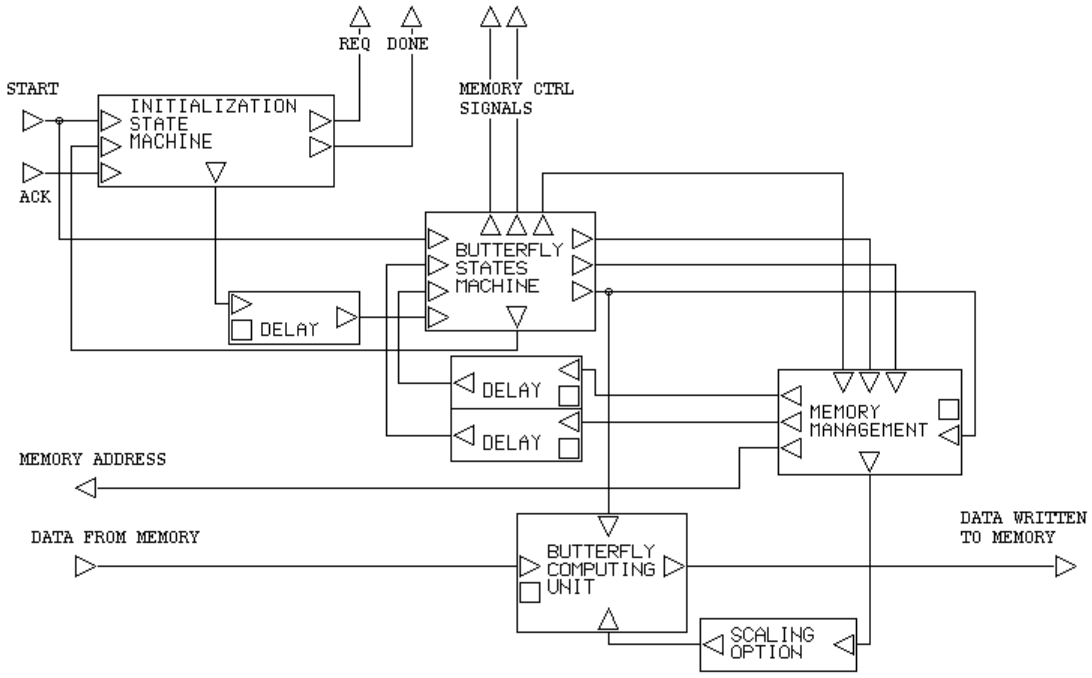


**Figure 4.2.2 BDE Diagram for FFT Engine**

## FFT Address Generation

Consider the following 16 pt FFT done by Decimation in frequency.

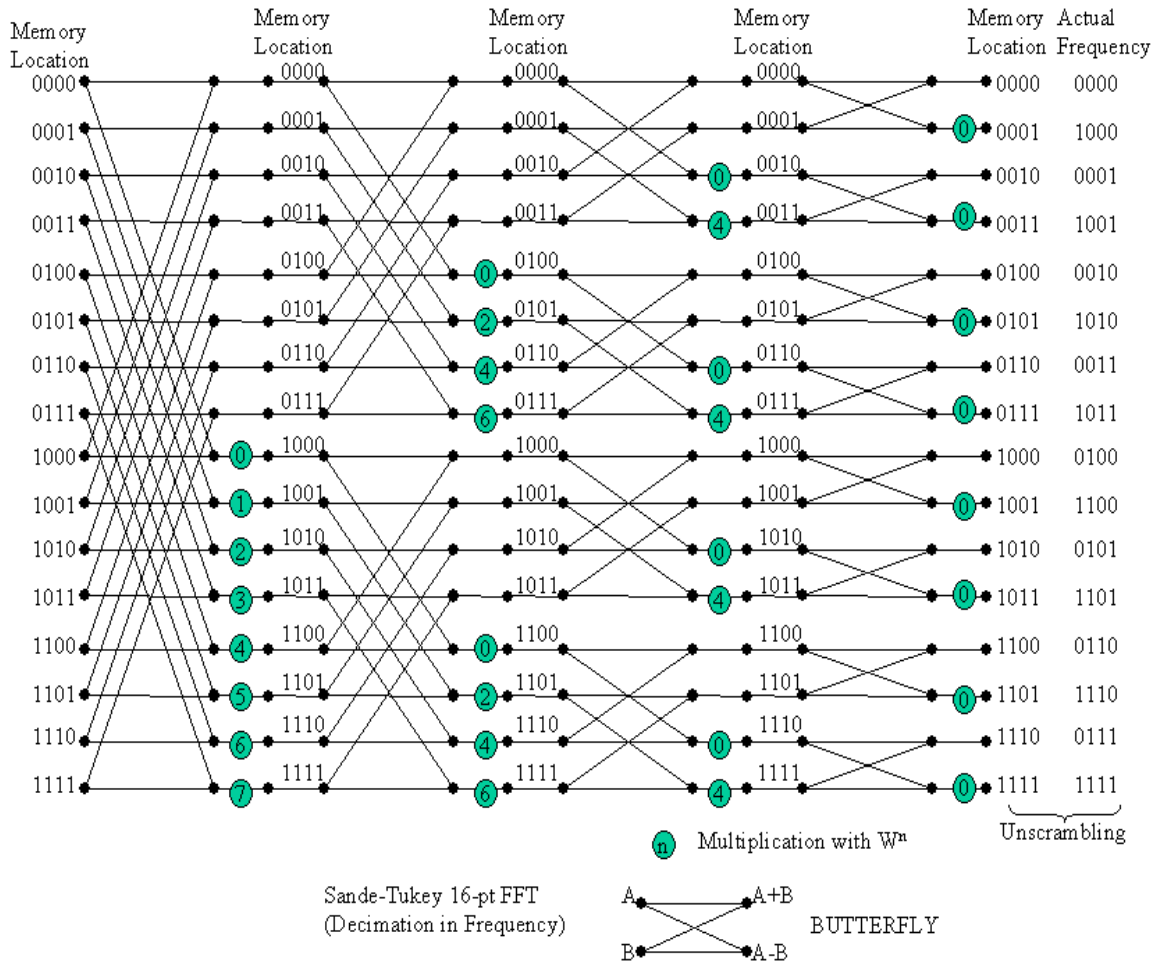


**Figure 4.2.3 16pt FFT by decimation in frequency**

Note that the memory read operations would be in the order as shown in the following Table 4.1:

| | |
|---|---|
| Stage 0 | 0000, 1000, 0001, 1001, 0010, 1010, …, 0111, 1111 |
| Stage 1 | 0000, 0100, 0001, 0101, 0010, 0110, 0011, 0111, 1000, 1100, …, 1011, 1111 |
| Stage 2 | 0000, 0010, 0001, 0011, 0100, 0110, 0101, 0111, 1000, 1010, …,1101, 1111 |
| Stage 3 | 0000, 0001, 0010, 0011, 0100, …, 1110, 1111 |

**Table 4.1 Order of Memory Read Operation**

Comparing with the regular 4 bit counter output, it can be seen that the address bit 3 (using convention 3:0 -> MSB:LSB), is identical to the LSB of the counter for stage 0, address bit 2 is identical to the LSB of the counter for stage 1, address bit 1 is identical to LSB of the counter for stage 2, and address bit 0 is identical to the LSB of the counter for stage 3. This trend makes it an easy solution for hardware designing, since this just maps to reordering of the lines of a regular up counter, depending on the stage. The above statements can be rephrased as: LSB of the counter output is inserted into the (M-2) MSBs of the counter output at the (M - stage)$^{th}$ bit location, to get the read address (For the N=16 pt FFT case, M = $\log_2(N)$ = 4, stage runs from 0 to 3). This can be generalized in BDE, by exporting the value of N and making the bitwidth of the counter to be log2(N). The block diagram of address generation is shown in the Figure 4.2.4 below.

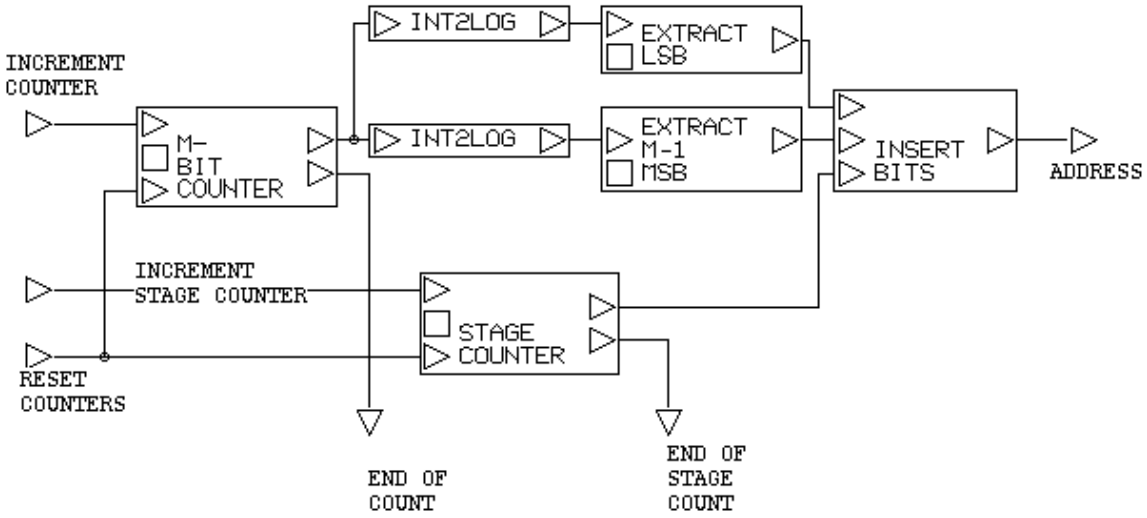


**Figure 4.2.4 FFT Data Address generation**

# 5 Design and FPGA Implementation of SAR Signal Processor

This chapter describes the design and implementation of SAR Signal Processor on FPGA. It describes, in brief, certain target-architecture specific optimizations done to enhance the overall performance of the Processor.

## *5.1 Analysis of SAR Raw Data*

The SAR raw data used for the design and implementation is CEOS level 1 data (see Appendix D) with minimum processing done on it. The data is raw in the sense that it is only downconverted and compared to the inphase reference and reference shifted by $90^o$. This work aims at processing this I and Q data, to get an image.

One data set file has one scene (100 km x 100 km), i.e. 8 patches, but this thesis work is targeted to only one patch, 4096x4096 data pixels. The data is 5-bit real and 5-bit imaginary padded with 3 '0' bits to make each real and imaginary part, a full byte data. The raw data requires computation of Doppler frequency at each sample. This forms a part of preprocessing of the raw data, which is assumed available to the actual signal processor (which is range compression, range migration and azimuth compression).

## *5.2 Design Partitioning*

The rectangular algorithm atleast implies that the order of processing would be Range Compression, Range Migration and last Azimuth Compression. To perform Compression in frequency domain is advantageous, and so FFT forms an important stage too. Range Compression is carried out in frequency domain by first carrying out FFT of the data, followed by range scaling the result and finally carrying out IFFT. One can consider pipelining the above 3 parts -FFT, scaling and IFFT across PEs to gain continuous output after initial setup latency. However, the success of this method is dependent on the next stage to follow. Since the next stage is Range Migration in azimuth frequency domain, one has to wait till the whole patch is range compressed, to perform a corner turn on the range compressed results, and then perform an FFT on the data along the other direction.

Thus, this now leaves us with evaluating any possible gains in pipelining the stages of Range Processing viz. FFT, Range Scaling in frequency domain and IFFT. Now in terms of throughput, FFT/IFFT will not provide any continuous output until its processing reaches the last stage of butterfly. For such large 4096 samples' line, latching all the values on FPGA is also not a feasible solution. (If number of samples in a line are, say, 128 then they could possibly be stored on FPGA until all processes involving these samples are not complete, for e.g. a complete Range processing for this line of 128 samples.)

Thus the best implementation for the WILDFORCE board architecture would be a data parallel approach, where same stage of SAR signal processor is loaded on the FPGAs and they all process the data until all 4096 lines are not processed.

Timing Issues: Partitioning of design should also take into consideration the % usage of FPGAs to implement the signal processor stage, since the place and route may not be able to provide a good clock speed for densely used FPGAs.

The logical partitioning which was implemented is shown in the Figure 5.2.1 below. The blocks shown in the bounding box, labeled "Processes on FPGA" are all designed as separate units and loaded on FPGAs as and when required.

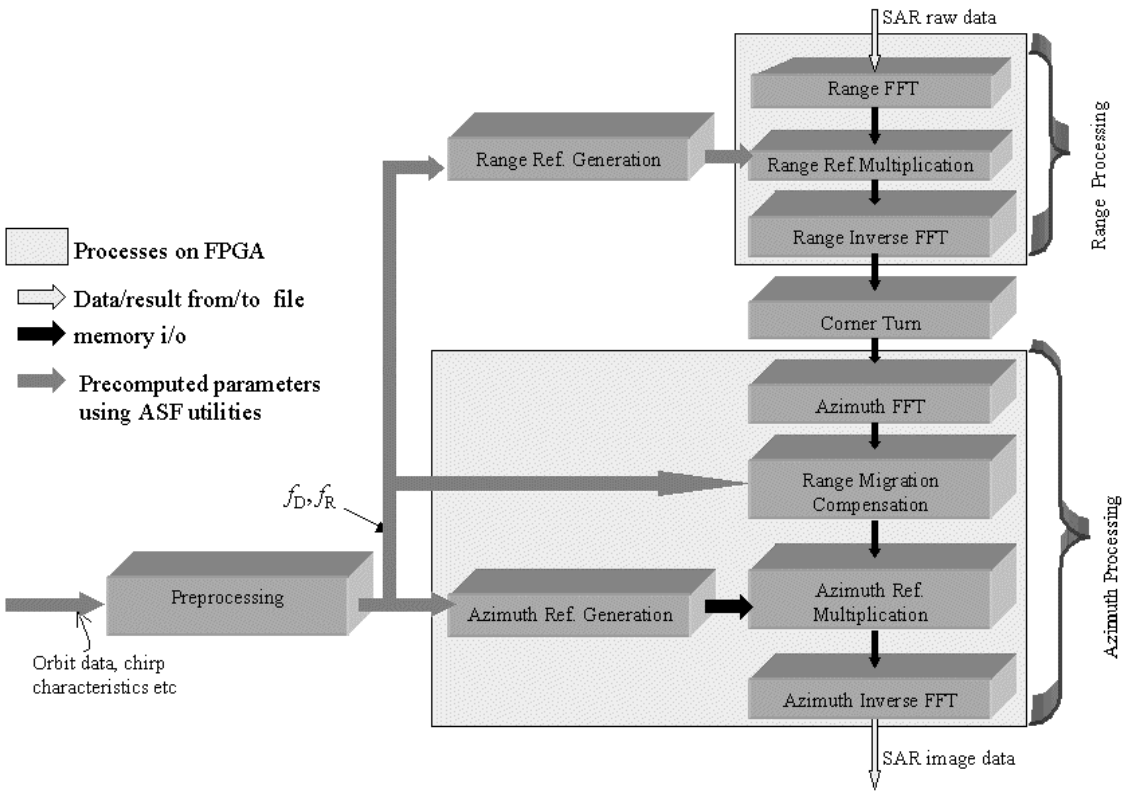


**Figure 5.2.1 SAR Signal Processing Design Partitioning**

Having now decided tentatively upon the design modules to be processed, one need to determine how would the interaction between host and PE be managed.

## *5.3 HOST-PE INTERACTION MODEL*

The host can communicate with the WILDFORCE board using 'C' API interface. Using these APIs it can reconfigure FPGAs, read/write to memory/FIFO and acknowledge interrupts.

For the SAR signal processor, after having designed each individual module, formed after deciding upon the way the design shall be partitioned, an effective and efficient interaction model between the PE(s) and Host needs to be designed. Considering the WILDFORCE architecture and its available 'C' APIs, the following model was thought of and designed.

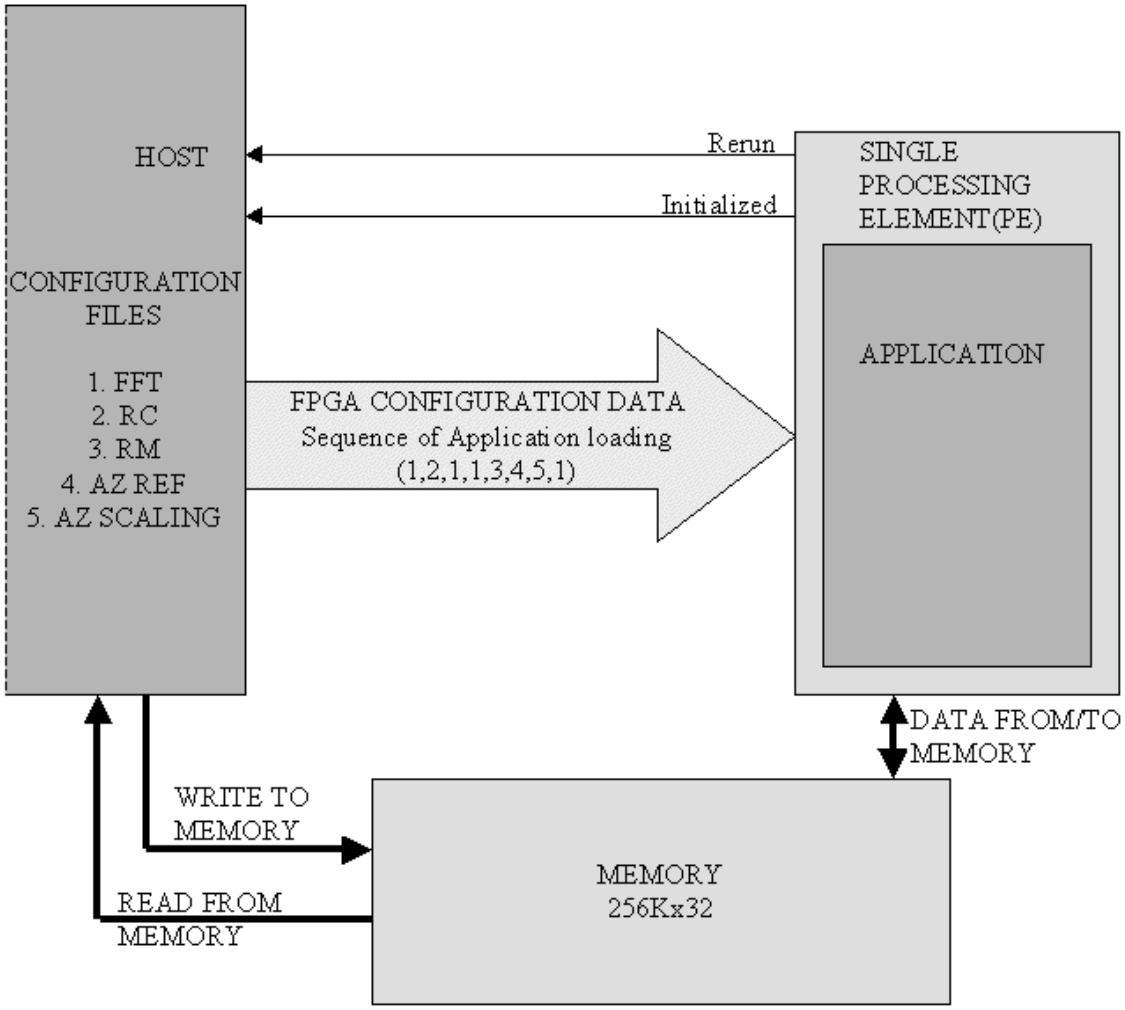


**Figure 5.3.1 HOST-PE Interaction Model**

The exchange of state information between the host and PE is carried by the pair of interrupts and interrupt acknowledgements. The number of interrupts is indicative of how many lines have been processed in a particular stage of SAR signal processing. Some further detailed information about this interaction is shown in the form of a flowchart in Figure 5.3.2 below.

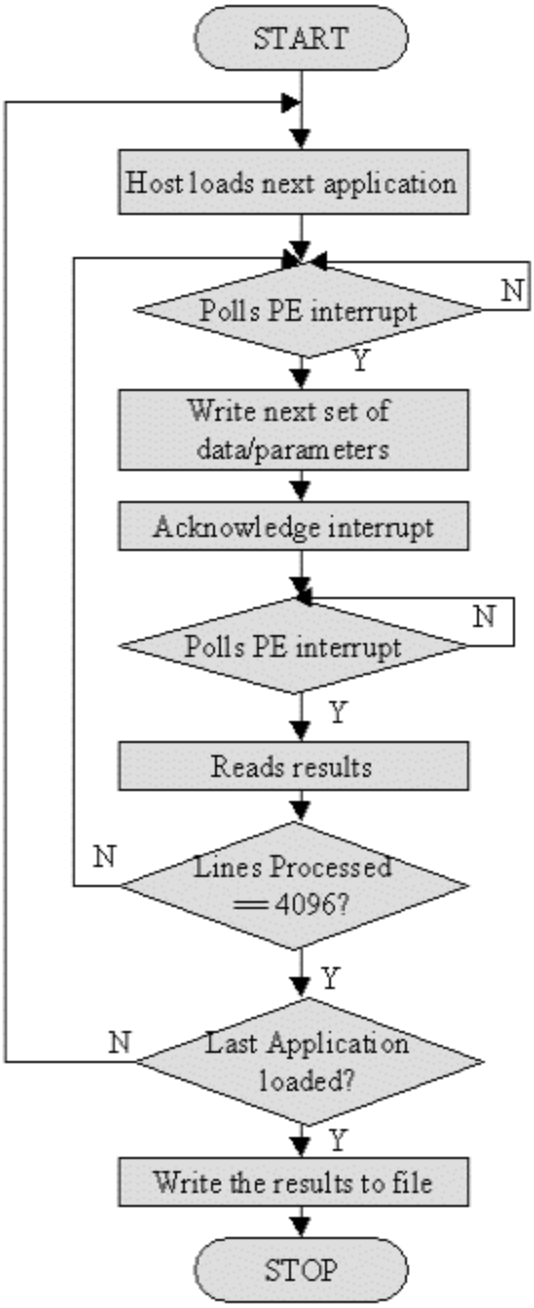


**Figure 5.3.2 Flowchart for Host-PE interaction**

The host loads the first application and waits for PE to interrupt, which would mean to the host that the PE is properly initialized. The host now writes the first set of data and parameters and acknowledges the interrupt it had received.

Host now waits for another interrupt from the PE to indicate completion of processing of first set of data. On receipt of this interrupt, host reads the results from the memory,

checks to see if all lines have been processed for the concerned application and if not then next batch of data/parameters is written on the memory. The host also acknowledges the interrupt. This process continues until the last batch of data/parameters is processed.
After the last batch of data is processed, the host checks if there are any more applications to be loaded on the PEs. If so then it loads the application, waits for the 1st interrupt indicating initialization-complete by the PE, writes 1st set of data/parameters onto the memory and acknowledge the interrupt.
At the end of the last interrupt, the results are written back to a file.

The implementation of various modules is discussed in detail. Since FFT is an important DSP algorithm, its implementation has already been dealt in a separate chapter 4.

### *5.4 Range Scaling*

Range Compression Stage is carried out in frequency domain. So the data is first Fourier transformed along each azimuth line. The range reference function is basically the linear FM pulse replica. Therefore, the number of non-zero values in the reference function in the time domain is equal to the number of samples in a chirp signal. This is equal to 703 for the case of ERS-1 satellite (reference_Length = pulse_duration * sampling_rate = 37.1 μs * 18.96 MHz = ~703). Then the 4096 pt FFT is performed on this reference function to get a reference function in the frequency domain. A windowing function (Hamming Window) is applied to arrive at a more practical matched filter implementation to perform the sidelobe reduction. The Memory model for this range compression is similar to the memory model for FFT. The 1st 4096 location stores the range reference function (in the frequency domain) and the 60 lines of data are stored from the 5th to 64th line location in memory. The 2nd through 4th line location in memory are unused.

The Range Compression scaling is performed as a separate module. It is a complex multiplication of the signal with the reference function. The top-level design unit in BDE for Range Compression is shown in Figure 5.4.1 below.

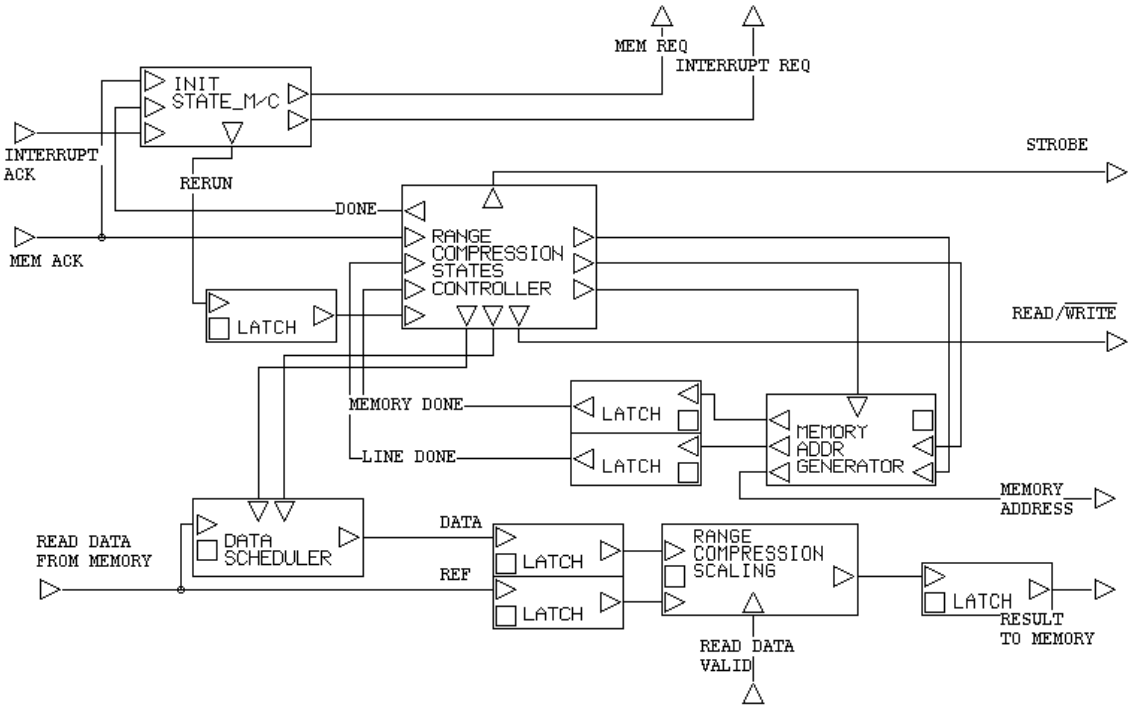


**Figure 5.4.1 BDE top level diagram for Range Compression**

After this scaling is performed, the scaled signal data is inverse Fourier transformed using IFFT, to get the data back into the time domain.

The Corner Turn process is carried out on the host computer, because of the limitation of the onboard memory on WILDFORCE. The azimuth FFT is carried out on the transposed data and the resultant data is then corrected for range migration followed by azimuth compression.

## *5.5 Range Migration*

As discussed before, for correcting the Range Migration, the parabolic trajectory of range shifts needs to be calculated. This implies that to calculate actual echo for a particular sample, the offset (range shift) is obtained by evaluating the parabolic trajectory for that range bin.

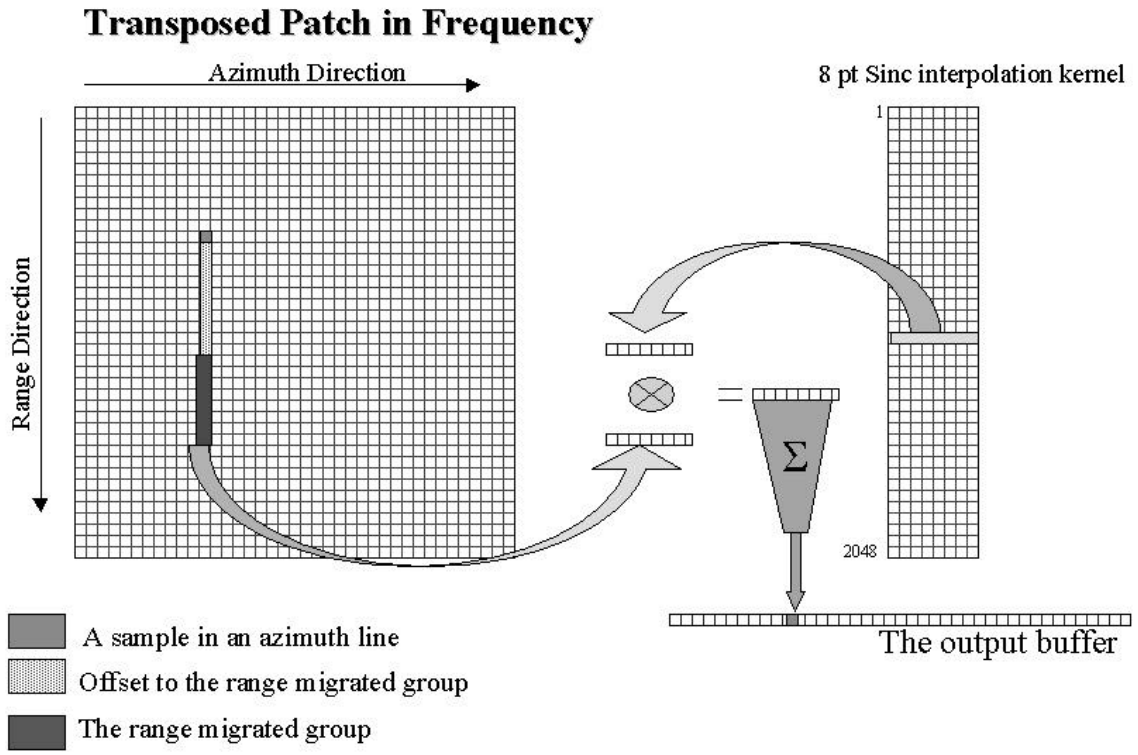


**Figure 5.5.1 Range Migration Correction Algorithm**

The Range Migration problem may require correction in both time and frequency domains. However, for small range migration, with beam squint angle at most a fraction of a degree, it may not be necessary to do any time domain range migration correction and all the correction can be carried out in the Doppler frequency domain. [Curlander et. al.] In fact, for C band system like ERS-1 radar, the time domain correction is not needed. The Doppler history gives the information about the offset to the range bin carrying the information for the particular Doppler frequency. The offset, which is obtained, will not usually be an integral number and so a simple polynomial interpolation using 8 values is performed. This interpolation and Doppler Domain Range Migration Correction is shown in the above Figure 5.5.1.

### Implementation Details

According to the analysis and derivation in [Levanon88] the offset for satellites like ERS-1 would not be more than a few range bins. Hence, a special cache is designed to hold the most probable data values, which the offset will point.

- Cache Design

To accommodate an 8-point interpolation kernel scaling of the data and to allow for possible range migration to some 8 pixels, a cache with atleast 16 location (32 bit wide) was required. The functional diagram for the designed cache is shown in the Figure 5.5.2 shown below.

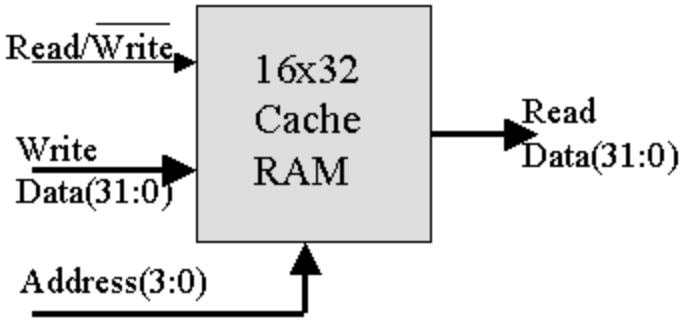


**Figure 5.5.2 Functional Diagram of Cache (16 WORDs) Design**

The implemented Range Migration Correction unit in BDE is shown in the Figure 5.5.3 below.

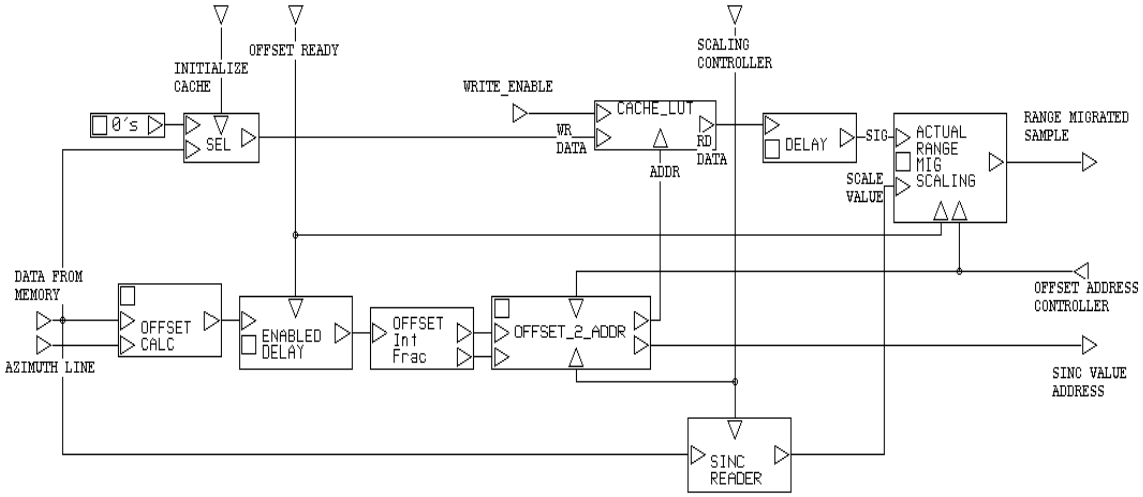


**Figure 5.5.3 Range Migration Correction Unit**

The calculated offset is saved through enabled latch to perform 8-pt sinc interpolation scaling for this sample. Appropriate integer and fractional bits are converted into addresses for CACHE, and sinc, which is in memory. The sinc values read from memory are used to compute corrected value of Range Migration sample. This is performed in "ACTUAL RANGE MIG SCALING" block.

### *5.6 Azimuth Reference Function Generation*

The Doppler Centroid, $f_D$, and Doppler rate, $f_R$, are assumed to be available either from the preprocessing of the image file or it is already available along with the raw data. In either case, the Doppler frequency for each range bin is just a value of a polynomial using $f_D$ and $f_R$. Now azimuth reference function is just a particular way of reading a sine cosine table, and hence the FFT module is merged with this module to get the reference function directly in the frequency domain. This would avoid the full extra cycle of loading the memory for a separate FFT. The Doppler frequency, normalized with respect to pulse repetition frequency is read in as a parameter. This normalization saves a large number of hardware expensive multiplications. Similarly scaling by $2*\pi$ is also performed in order to save all sorts of extra computations and work around with bit shifts is carried out.

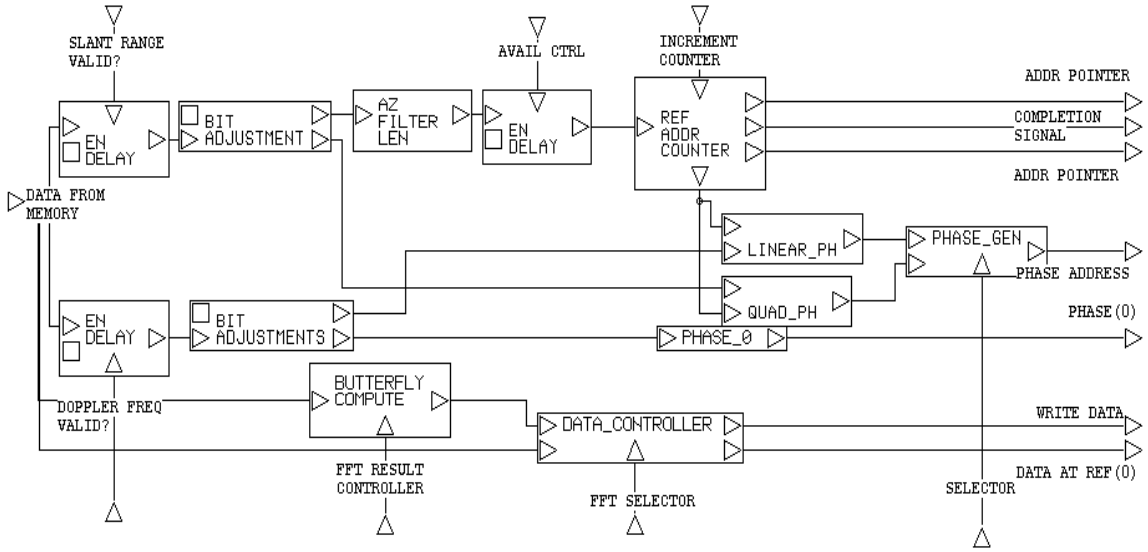


**Figure 5.6.1 BDE diagram for Azimuth Reference Calculation Design**

### *5.7 Azimuth Compression*

Azimuth Compression would be a frequency domain implementation of the matched filter. So the range migrated data samples are multiplied with the complex conjugate of the azimuth reference function, calculated in previous stage. The output is further weighed with azimuth weighting function to suppress sidelobes.

### Implementation Details

Since the azimuth reference function is different for each range bin, there is one reference value to be read for each data value. This leads to half the throughput. The memory can now be used to process only 30 data lines instead of 60 data lines as done with other modules. The other 30 data lines are used by the azimuth reference function. The Figure 5.7.1 shows the top-level design in BDE for this stage.

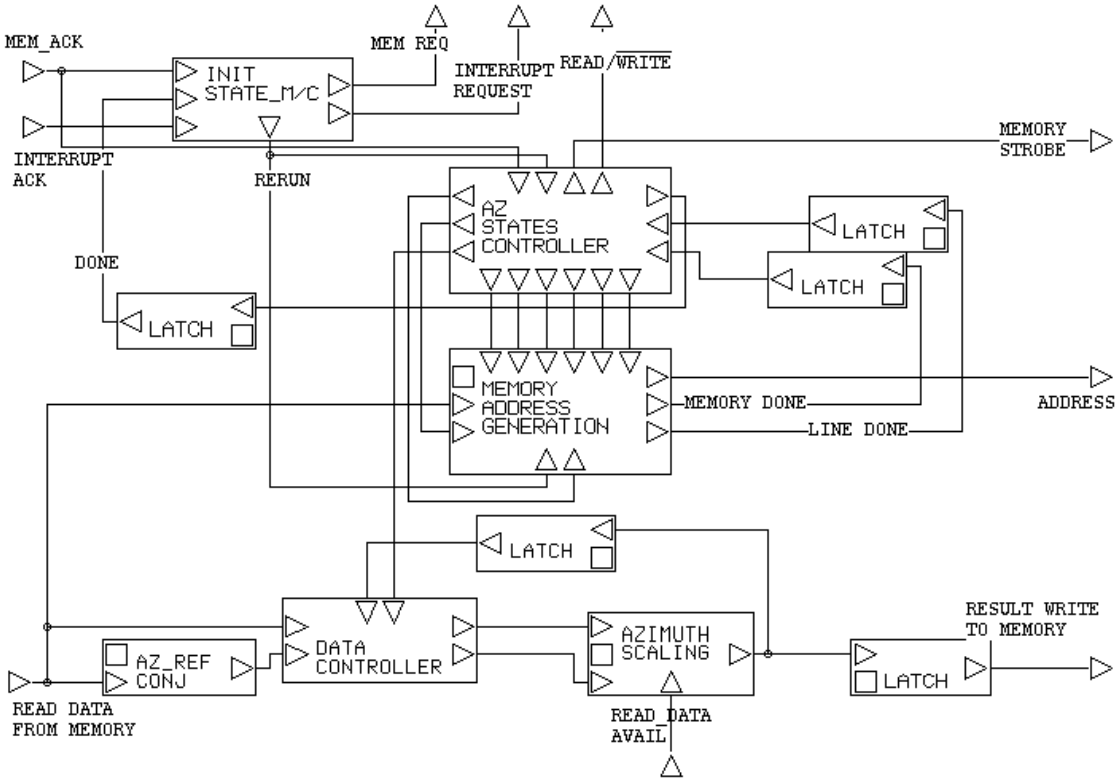


**Figure 5.7.1 BDE diagram for top level design of Azimuth Scaling**

The "AZ_STATES CONTROLLER" is a finite state machine (FSM) generating block, which emits appropriate control signals for all other blocks. The memory address generation toggles the address lines from data storing location, azimuth reference storing location, azimuth weighing value storing location and result write location. The data controller schedules the numbers to be fed into the "AZIMUTH SCALING", which has at its core a complex multiplication block.

After azimuth compression is performed, the image so obtained is inverse FFT'ed to bring back to the time domain, and then corner turned to rearrange the samples. To compare with the ASF image, which is multilooked, the complex image obtained through FPGA is then multilooked using ASF's utility.

Thus having designed all the modules through BDE/FLASH and optimized for speed, their device utilization summary is provided next.

### *5.8 Figures of Implementation*

Table 5.1 to Table 5.5 gives the FPGA place and route tools results for the various modules of SAR signal Processor.

| **FFT/IFFT** | PE0 | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|---|
| CLBs | 1279 (40%) | 1258 (40%) | 1278 (40%) | 1278 (40%) | 1258 (40%) |
| 4 i/p LUT | 2228 (35%) | 2182 (34%) | 226 (35%) | 226 (35%) | 2182 (34%) |
| 3 i/p LUT | 96 (3%) | 68 (2%) | 91 (2%) | 91 (2%) | 68 (2%) |
| Max. Clk (MHz) | 13.82 | 16.430 | 13.06 | 13.52 | 16.430 |

**Table 5.1 Device Utilization summary for FFT/IFFT**

| **RC-scaling** | PE0 | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|---|
| CLBs | 1175 (37%) | 1188 (37%) | 1176 (37%) | 1176 (37%) | 1188 (37%) |
| 4 i/p LUT | 1952 (31%) | 1976 (31%) | 1952 (31%) | 1952 (31%) | 1976 (31%) |
| 3 i/p LUT | 173 (5%) | 172 (5%) | 175 (5%) | 175 (5%) | 172 (5%) |
| Max. Clk (MHz) | 14.3 | 14.44 | 14.32 | 14.32 | 14.11 |

**Table 5.2 Device Utilization summary for Range Scaling**

| **RM** | PE0 | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|---|
| CLBs | 1920 (61%) | 1941 (61%) | 1920 (61%) | 1920 (61%) | 1941 (61%) |
| 4 i/p LUT | 3050 (48%) | 3085 (49%) | 3045 (48%) | 3045 (48%) | 3085 (49%) |
| 3 i/p LUT | 431 (13%) | 446 (14%) | 431 (13%) | 431 (13%) | 446 (14%) |
| Max. Clk (MHz) | 9.1 | 10.623 | 9.713 | 9.713 | 10.623 |

**Table 5.3 Device Utilization summary for Range Migration**

| AZ_ref | PE0 | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|---|
| CLBs | 2014 (64%) | 1953 (62%) | 2025 (64%) | 2025 (64%) | 1953 (62%) |
| 4 i/p LUT | 3456 (55%) | 3326 (53%) | 3453 (55%) | 3453 (55%) | 3326 (53%) |
| 3 i/p LUT | 157 (5%) | 159 (5%) | 172 (5%) | 172 (5%) | 159 (5%) |
| Max. Clk (MHz) | 10.223 | 10.262 | 8.953 | 10.262 | 8.953 |

**Table 5.4 Device Utilization summary for Azimuth Reference Generation**

| AZ_Scaling | PE0 | PE1 | PE2 | PE3 | PE4 |
|---|---|---|---|---|---|
| CLBs | 1590(50%) | 1602 (51%) | 1591 (50%) | 1591 (50%) | 1602 (51%) |
| 4 i/p LUT | 2656(42%) | 2665 (42%) | 2656 (42%) | 2656 (42%) | 2665 (42%) |
| 3 i/p LUT | 145(4%) | 138 (4%) | 145 (4%) | 145 (4%) | 138 (4%) |
| Max. Clk (MHz) | 14.35 | 13.41 | 13.177 | 13.177 | 13.41 |

**Table 5.5 Device Utilization summary for Azimuth scaling**

Table 5.6 gives the information about the number of lines that could be processed with one full load of memory. This is directly proportional to the memory size and the numbers in the table are corresponding to 256K WORD memory. (4 lines of 4096 memory location are used in all modules to store various precomputed parameters.)

| **Modules** | FFT | RC | RM | AZ_REF | AZ_C |
|---|---|---|---|---|---|
| **#lines/reload** | 60 | 60 | 60 | 60 | 30 |

**Table 5.6 Throughput (#datalines_processed/reload)**

# 6 Results and Performance Evaluation

The SAR signal processor on FPGA design was tested directly on actual data obtained from ASF. The data is obtained on 24th October 1995. The data consists of the scene indicated by scene indicator(see Appendix E) E122361290S0C014 and the scene center's latitude and longitude are 63.9282$^{o}$ and -145.60558$^{o}$ respectively. This information is also available in the Dataset Summary record present in the metadata file.

## *6.1 Images*

Following is the image obtained from running the *aisp* software from ASF on a Linux machine, 350 MHz Pentium II, 256 MB RAM.

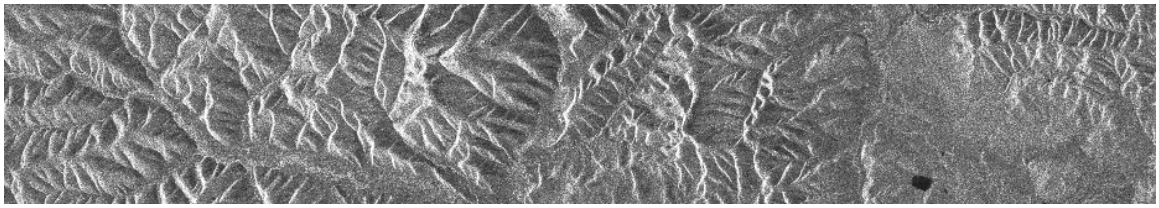

**Figure 6.1.1 Original image using *aisp* software**

The following image is generated through the FPGA processing of SAR signal processor. Note that the result does not have the same amount of resolution and hence contrast as the above image. This can be attributed to loss of accuracy in a fixed-point implementation.

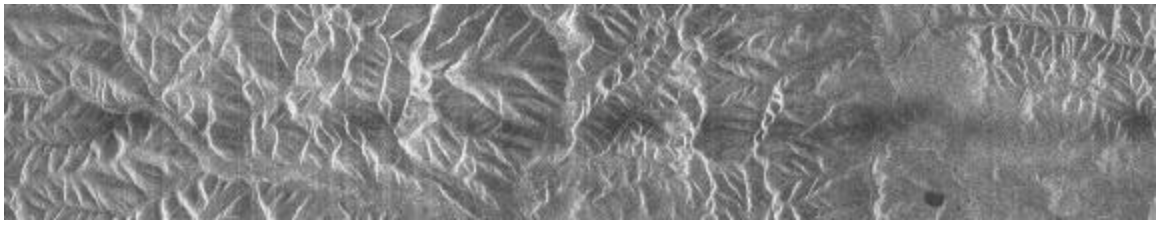

**Figure 6.1.2 Image obtained from SAR implementation on FPGAs**

The following table, Table 6.1 gives the number of cycles required to process one batch of memory load. The number of cycles is based on the finite state machine being used. Note that these values are rounded off to the nearest hundred, since the exact time in interrupt exchange between PEs and host is not a fixed value. For the maximum clock speed available, it also gives the time to process one batch of memory.

| **Modules** | **FFT** | **RC** | **RM** | **AZ_REF** | **AZ_C** |
|---|---|---|---|---|---|
| **Cycles/memory** | 10322000 | 921700 | 4671200 | 10679000 | 1843300 |
| **Max Clk (MHz)** | ~13 | ~14 | ~10 | ~10 | ~13 |
| **Time (sec)/memory** | 0.794 | 0.066 | 0.467 | 1.068 | 0.142 |
| **Time (sec)/patch**[5] | 11.12 | 0.924 | 6.538 | 14.95 | 3.976 |

**Table 6.1 Time to process by a module**

To calculate the Time/patch the following procedure is used:

Total lines: 4096

#lines/reload: from Table 5.6

#reloads = (Total lines)/( #lines/reload) = 4096/60 = ~69… for FFT, RC, RM, AZREF

= 4096/30 = ~137… for AC

#PEs in parallel: 5

#reloads/PE = (#reloads) / (#PE) = 69 / 5 = ~14 for FFT, RC, RM, AZREF

= 137/5 = ~28 for AC

Time/patch = (Time/memory) * (#reloads/PE)

Therefore, the time to bare computations would be,

Time = 4*FFT_Time + RC_Time + RM_Time + AZREF_Time + AC_Time

= 4(11.12) + 0.924 +6.538 +14.95 +3.976

= 70.87 seconds

[5] With all 5 PEs working in parallel, patch=4096 lines, while for each module, a single load of memory will hold lines as given by Table 5.6

The timing reports of all the stages are given in Table 6.2. The timing values in Table 6.2, are averages of 3 runs. Note that the improvement for smaller processes like RC and AC is not available on upgrading from 1 to 3 to 5 PEs. There is a significant improvement in timings for the modules using FFT. This behavior can be quantified to get an optimal value of PEs for the given overhead of loading and reloading the memory through the APIs.

| **Time (sec.)** | 1 PE no DMA | 1 PE DMA | 3 PEs no DMA | 3 PEs DMA | 5 PEs no DMA | 5 PEs DMA |
|---|---|---|---|---|---|---|
| R.FFT | 55.10 | 44.93 | 30.26 | 19.65 | 24.32 | 13.5 |
| RC | 17.66 | 5.53 | 18.77 | 5.45 | 19.84 | 5.37 |
| R.IFFT | 56.97 | 46.76 | 31.47 | 20.90 | 25.08 | 14.28 |
| A.FFT | 54.86 | 44.66 | 29.44 | 18.79 | 23.09 | 12.16 |
| RM | 46.61 | 36.34 | 24.14 | 13.58 | 20.73 | 9.93 |
| AZREF | 82.35 | 72.9 | 41.47 | 30.94 | 30.89 | 20.24 |
| AC | 24.35 | 12.66 | 25.39 | 5.97 | 26.56 | 5.49 |
| A.IFFT | 56.98 | 46.79 | 31.50 | 20.94 | 25.17 | 14.34 |

**Table 6.2 Timing Report for all 8 stages of SAR Processor**

Also, note that there is a significant improvement in using DMA. However, the efficiency of this implementation is not significantly better than the software implementation. In fact, except for the 5 PEs with DMA case, none provide better timings as compared to the software implementation.

Table 6.3 compares the time for actual computation with the software implementation timings. It clearly shows that this current design implementation is not a significant improvement to its software counterpart. However, note that the major drawback is the memory I/O time. The actual timings of calculation as shown in Table 6.1 clearly shows that the parallel processing feature of FPGAs offer the faster solution.

| Implementation | | Time for actual computation (sec)/ Total time[6] (sec) |
|---|---|---|
| Software - ASF's *aisp* utility | | 103/106.4 |
| **Using regular memory transfer** | | |
| #PEs | 1 | 394.9/402.34 |
| | 3 | 232.43/240.05 |
| | 5 | 195.69/203.32 |
| **Using DMA memory transfer** | | |
| #PEs | 1 | 310.57/318.13 |
| | 3 | 136.37/145.6 |
| | 5 | 95.32/102.56 |

**Table 6.3 Comparison of Time for implementation**

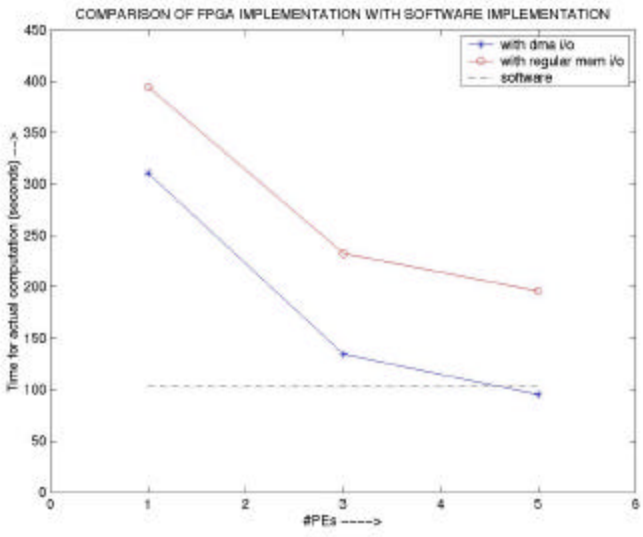


**Figure 6.1.3Comparison Plot of Time to process a patch**

[6] This time is the sum of time for actual computation, corner turn of the patch, and data scaling after reading raw data.

# 7 Conclusion and Future Work

This thesis demonstrates the design and FPGA implementation of certain algorithms of Synthetic Aperture Radar signal processor. It also shows how the same piece of silicon can be used efficiently by reprogramming it with the different stages of the processor one after the other. This work also shows the effectiveness of FLASH and BDE in implementing such a complex design.

The resultant SAR image shows a successful implementation of a prototype SAR signal processor on FPGA. The memory I/O is the bottleneck of this process. With a larger memory size of storage, the bare-computation time limit can be reached through actual implementation too.

## *7.1 Limitations*

The results that have been generated do not provide improved performance over software implementation because of some limitations in the implementation like:

- Large Memory I/O time
- Only 1MB of memory on each PE, resulting into large number of reloads.

## *7.2 Future Work*

This work has concentrated on developing a SAR signal processor with the basic modules. So some of the possible future work in this area can be:

- Incorporating various other stages of SAR image processing, like speckle reduction, geocoding, Interferometry, etc to work after this signal processor.
- To generate Doppler Centroid and Doppler rate quantities by on board calculation, instead of using precomputed values.
- Provide additional flexibility in the existing design, by allowing parameterized image size, different FFT size for each stages etc.
- Higher Radix FFT can be implemented and compared to check for better performance.

- Implementing this work with floating point representation, to explore the advantage of larger dynamic range of floating point over fixed point representation. However, the simple additions would be a bit more complex with floating point representation.

# Appendices

## *Appendix A: ERS-1 satellite characteristics*

| | |
|---|---|
| Orbit | |
| Altitude | 785 km |
| Inclination | 98.516° |
| Ground Track velocity | 6.628 km/s |
| Instrument | |
| Frequency | C-Band (5.3 GHz) |
| Wavelength | 5.66 cm |
| Pulse Repetition Frequency | 1640-1720 Hz |
| Pulse Length(BW) | 37.1 μs (15.5 MHz) |
| Polarization | VV |
| Antenna Size | 10 X 1 m |
| Antenna 3 dB Width | |
| Range | $6.0^{o}$ |
| Azimuth | $0.3^{o}$ |
| Image | |
| Swath Width | 100 km |
| Max Resolution Range X Az. | 12.5 X 12.5 m |
| Resolution @ #looks | 30 @ 4 |
| System | |
| Look Angle | Right 20.355° |
| Incidence Angles, Mid | 19.35-26.5°,23° |
| Footprint Range X Azimuth | 80 X 4.8 km |
| Doppler BW | 1260 Hz |
| Coherent Integration Time | 0.6 s |
| Windowing | Hamming |
| Pulse Compression Ratio | 580:1 |
| Range Sample Rate | 18.96 Mb/s |
| Maximum Data rate | 105 Mb/s |
| Quantization | 5 bits per sample |

**Table A.1 Orbit parameters for ERS-1**

(Source: ASF Scientific SAR User's Guide)

### *Appendix B: ASF CCSD Data-set Format*

| Product Name | Bytes per Record by Number of Records, including CEOS | Actual Data Samples | Data Type | Byte Order | Record Order (w.r.t. time) |
|---|---|---|---|---|---|
| ERS-1, Computer-Compatible Signal Data | 11456 x 26626 | 26625 records of 5632 samples | 2 Bytes (8 bits Real + 8 bits Imaginary) (5I + 5Q significant bits) | Real, Imaginary | First to last |

**Table A.2 ASF CCSD Data set Format**

### *Appendix C: ASF CCSD Data files detail*

The data product has two files. One containing the data for the image and the other containing information about the data, the image, and information to go with the image. The data file, one of the files, contains nothing but image data. It is simply a huge chunk of continuous data. (It has an extension .D) The other type of file is the metadata file, which contains all the gathered information about the data. (Extension ".L" with the same basename as ".D" file). The metadata file contains important information for viewing the image, manipulating the image, processing the image, and archiving the image. More information is available at ASF's CEOS specifications page. First, the metadata file contains the image information such as the range (width) and azimuth (length or height) of the image. Along with the image information, the metadata contains information such as the position and orientation of the satellite at the time, the image data was gathered.

### *Appendix D: CEOS Image Data*

CEOS Data is one of the main types of data that the Alaska SAR Facility (ASF) produces. There are three main levels of CEOS data. The first, known as CEOS Level 0 data, has two main formats. The first format is the raw stream of data transmitted from the satellite. This data is simply the unformatted jumble of bits received from the satellite. The second format of CEOS Level 0 data is the formatted stream of satellite data. This has the sync codes removed and is byte aligned for ease of integration into a computer system. CEOS Level 0 data must be processed further to yield useable data products such as images and other derived data.

This further processed data is known as CEOS Level 1 data. CEOS Level 1 data is mainly in the form of images that are derived from CEOS Level 0 data. The level of processing done on the data varies from the raw image, such as what the satellite "sees," to flatten out geocoded images. This processing is done by the different tools developed and maintained at ASF.

Further processing of these images, results in data types known as CEOS Level 2 data. Such things are usually the result of multiple images and their comparison. Data such as ice motion and Interferometry are examples of CEOS Level 2 data.

### *Appendix E: Decoding the scene indicator*

Consider the scene indicator: E122361290S0C014 (data file name also is the same character appended by .D)

Where: **E1** ---- for ERS-1

**22361** ---- Orbit Number (ranges from 00000 to 99999)

**290** ---- Fixed frame numbering scheme relative to ascending node

**S** ---- indicates Slant range (Projection type of Data)

**0** ---- for Data Pixel Spacing = 6.25

**C** ---- For CCSD data type

**14** ---- Version Number

## *Appendix F: List of Acronyms*

| | |
|---|---|
| ASF | Alaska SAR Facility |
| ACS | Adaptive Computing Systems |
| CCSD | Computer Compatible Signal Data |
| CEOS | Committee on Earth Observation Satellites |
| DARPA | Defense Advanced Research Projects Agency |
| ERS | European Remote Sensing Satellite |
| ESA | European Space Agency |
| FPE | Functional Programming Environment |
| PRF | Pulse Repetition Frequency |
| SAR | Synthetic Aperture Radar |

### *References*


**[ASPA]** ASF SAR Processing Algorithm, June 2000. http://www.asf.alaska.edu/reference_documents/datacenters_references/sar_processing.html

**[ACS_Report2000]** Technical Report ITTC-FY2001-TR-13530-07, Code Listing and BDE diagrams for SAR signal processor on FPGAs, Information Telecommunication & Technology Center, University of Kansas, June 2000.

**[Andraka]** Andraka R., Berkun A., *'FPGAs Makes a Radar Signal Processor on a Chip a Reality'*

**[Bamler93]** Bamler R., Breit H., Steinbrecher U., Just D., *Algorithms for X-SAR Processing,* Remote Sensing Digest 1993.

**[Carrara95]** Carrara W.G., Goodman R.S., Majewski R.M., *Spotlight Synthetic Aperture Radar Signal Processing Algorithms,* Artech House, 1995.

**[Curlander91]** Curlander J.C., McDonough R.N., *Synthetic Aperture Radar: Systems and Signal Processing,* John Wiley & Sons, New York, NY, 1991.

**[Dandalis97]** Dandalis A., Prasanna V. K., *Fast Parallel Implementation of DFT using Configurable Devices,* International Workshop on Field Programmable Logic and Applications, 1997

**[Levanon88]** Levanon N., *Radar Principles,* Wiley-Interscience Publications, 1988.

**[Muehring97]** Muehring J. T., Thesis Texas Tech University, *'Optimal Configuration of a Parallel Embedded System for Synthetic Aperture Radar Processing'*

**[Pridgen95]** Pridgen J., Jaffe R., Kline W., *RASSP Technology Insertion into the Synthetic Aperture Radar Image Processor Application,* Second Annual RASSP Conference, 1995.

**[Pryde94]** Pryde G.C., Beckett K.D.R., Delves L.M., Oliver C.J., White R.C., *Design of a real-time high quality SAR processor,* SPIE 94

**[RASSP_SAR]** *SAR CASE STUDY on RASSP* , June 2000 http://www.atl.lmco.com/projects/rassp/RASSP_legacy/casestudies/SAR/

**[SSUG93]** Alaska SAR Facility, *Scientific SAR User's Guide,* Coert Olmsted, July 1993

**[Walker96]** Walker J. S., *Fast Fourier Transforms*, 2nd Edition, CRC Press, 1996

**[Wanhammar99]** Wanhammar L., *DSP Integrated Circuits*, Academic Press, 1999

**[Wegmuller97]** Wegmüller U. and C. L. Werner, *GAMMA SAR processor and interferometry software*, Proceedings 3rd ERS Scientific Symposium, March 1997 http://florence97.ers-symposium.org/papers/, as of June 2000

**[Zuerndorfer94]** Zuerndorfer B., and Shaw G., *SAR Processing for RASSP Application*, Proceedings of 1st Annual RASSP Conference, 1994 pp 253-268