Journal of the KIMST, Vol. 20, No. 4, pp. 482-490, 2017
ISSN 1598-9127
DOI http://dx.doi.org/10.9766/KIMST.2017.20.4.482

Research Paper 센서 · 신호처리 부문

FPGA를 이용한 레이더 신호처리 설계

하창훈*,1) · 권보준2) · 이만규1)

1) 한화시스템㈜ 레이다 · PGM 연구소
2) 국방과학연구소 제3기술연구본부

Radar Signal Processor Design Using FPGA

Changhun Ha*,1) · Bojun Kwon2) · Mangyu Lee1)

1) *Radar · PGM Research and Development Institute, Hanwha Systems, Korea*
2) *The 3rd Research and Development Institute, Agency for Defense Development, Korea*

(Received 11 November 2016 / Revised 28 March 2017 / Accepted 30 June 2017)

ABSTRACT

The radar signal processing procedure is divided into the pre-processing such as frequency down converting, down sampling, pulse compression, and etc, and the post-processing such as doppler filtering, extracting target information, detecting, tracking, and etc. The former is generally designed using FPGA because the procedure is relatively simple even though there are large amounts of ADC data to organize very quickly. On the other hand, in general, the latter is parallel processed by multiple DSPs because of complexity, flexibility and real-time processing. This paper presents the radar signal processor design using FPGA which includes not only the pre-processing but also the post-processing such as doppler filtering, bore-sight error, NCI(Non-Coherent Integration), CFAR(Constant False Alarm Rate) and etc.

Key Words : Radar(레이더), Signal Processing(신호처리), FPGA, System Generator, CFAR(고정 오경보율)

1. 서 론

유도무기체계에 적용되는 전파고도계, RF 탐색기, 전자식 근접신관 등의 소형 레이더 장비는 유도탄 내부의 배터리 전원을 사용하며, 유도탄의 고기동 환경에 적용해야하기 때문에 소형화·경량화 및 낮은 소모전력 등이 요구된다. 반면에 최근 유도무기는 정밀도 향상을 위하여 복합기능, 고해상도 신호처리 등이 요구되어 빠른 연산속도를 갖는 신호처리기가 필요한 상황이다.

기존 레이더 장비 신호처리 방식은 다수의 DSP (Digital Signal Processor) 기반의 소프트웨어 신호처리 방식을 사용하였다. 이러한 소프트웨어 신호처리 방식은 개발이 용이하고 신호처리 알고리즘 수정·변경이 쉽지만, 소형·경량·저전력의 기본 요구사항을 만족하

* Corresponding author, E-mail: changhun.ha@hanwha.com

면서 다기능·고해상도 처리에 필요한 연산능력을 향상시키는 데는 한계가 있다.

최근 레이더 장비의 연산능력 향상을 위해 GPGPU (General-Purpose computing on Graphics Processing Units)을 이용한 신호처리기가 개발되고 있다. 하지만 GPU 프로세싱은 FPGA 비용대비 성능이 높지만, 소모전력 대비 성능이 낮아 소형·경량·저전력이 요구되는 장비 적용에는 부적합하다는 평가이다[1].

본 논문에서는 위와 같은 한계를 극복하고자, 기존의 레이더 소프트웨어 신호처리 방식을 FPGA 로직으로 설계 및 검증한 내용을 언급한다. 로직 개발은 신호 전처리와 후처리 로직으로 구분하여 진행하였고, 개발 툴은 Xilinx 사의 System Generator를 이용하였다. System Generator는 MATLAB Simulink 상에서 로직을 개발하기 때문에 기존 VHDL이나 Verilog에 익숙하지 않은 사람도 쉽게 신호처리 로직을 개발하고 검증할 수 있는 장점이 있다. Fig. 1은 System Generator를 기반으로 한 로직 검증 방법이다.

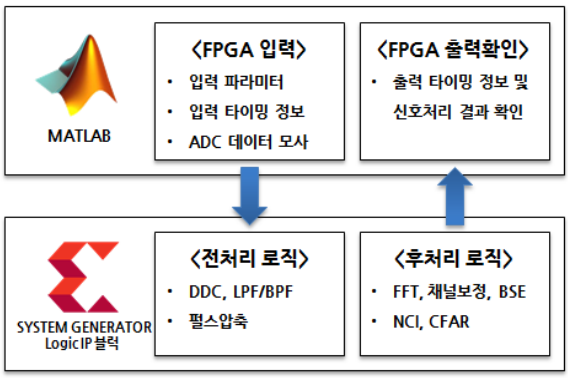

Fig. 1. Validation methode of signal processing logic

2. 레이더 신호 전처리 로직 구현 및 검증

본 연구가 적용되는 레이더 시스템에서는 운용 파형에 따라 두 종류의 독립적인 신호처리 경로를 거치는데 Fig. 2, 3은 각 신호처리 경로에 따른 전처리 과정을 나타낸다.

먼저 신호처리 경로 1의 경우에는 수신기 출력인 IF(Intermedia Frequency) 신호에 대한 주파수 해석만을 수행하는 처리 경로로 IF 신호를 받아 ADC를 수행하고 Fig. 2와 같이 I/Q 데이터 생성을 위한 디지털 하향변환(DDC, Digital Down Converter), 다운샘플링 및 디지털 필터링 연산을 수행한다.

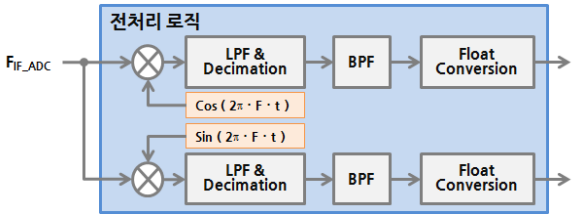

Fig. 2. Preprocessing flow of path 1

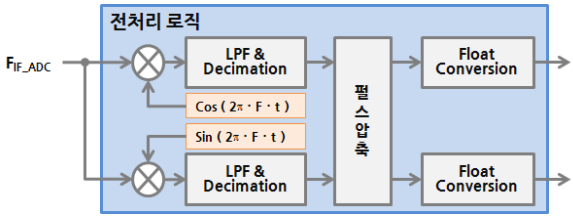

Fig. 3. Preprocessing flow of path 2

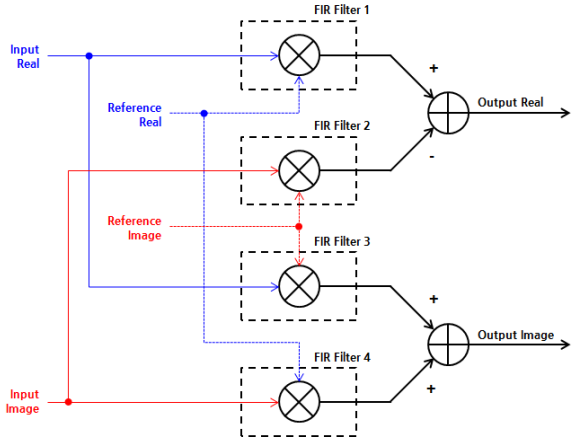

Fig. 4. Pulse compression concept using FIR filter

경로 2는 Fig. 3과 같이 변조 신호의 에너지를 모으고 반사 신호의 거리지연을 확인하기 위한 펄스압축 과정이 추가된다. 선형 주파수 변조 파형에 대한 펄스압축은 정합필터를 통해 계산되며, 이산 정합필터의 계산과정은 식 (1)과 같다[2].

$$g(n) = \frac{1}{N}\sum_{k=0}^{N-1} r(k)S^*(n-k) \quad (1)$$

여기서 g는 이산 정합필터의 출력, r은 수신신호, S^*는 기준신호의 complex conjugate이다.

정합필터를 FPGA로 구현하기 위해서, Fig. 4와 같이 기준신호를 필터계수로 사용하는 4개의 FIR 필터를 사용하였다. 펄스압축의 실수부결과는 FIR1과 FIR2의 차로, 허수부결과는 FIR3과 FIR4의 합으로 계산된다[3].

Fig. 5, 6은 System Generator에서 구현한 전처리 로직 블록도이다. Fig. 6에서 구현한 펄스압축은 실수부·허수부의 기준신호를 필터계수로 사용하는 두 개의 8채널(= Real/Imag 입력 × 4채널)의 FIR필터로 구현하였다.

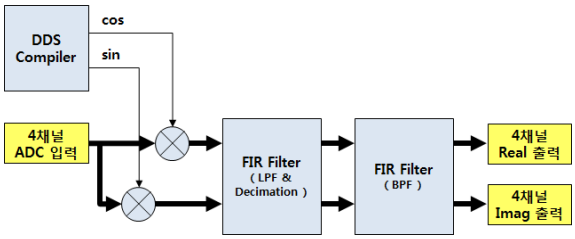

Fig. 5. Preprocessing logic of case 1

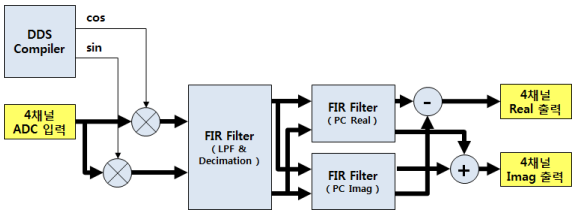

Fig. 6. Preprocessing logic of case 2

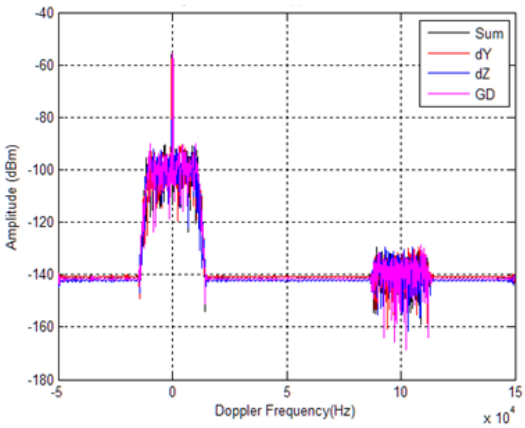

Fig. 7. Preprocessing results of case 1

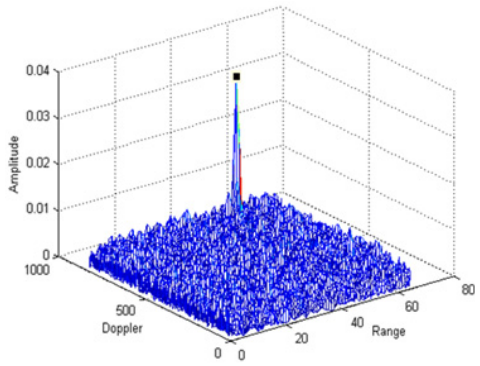

Fig. 8. Preprocessing results of case 2

개발한 FPGA 전처리 로직의 성능 확인을 위하여 MATLAB 시뮬레이션 결과와 비교해 보았다. Fig. 7, 8은 개발한 FPGA 전처리 로직 출력을 MATLAB에서 후처리(펄스압축·FFT) 한 결과이다. 비교결과 3.0E-6dB 미만의 계산차이를 보였는데, 이는 로직 내부에 고정소수점(Fixed Point) 연산을 수행한 후 최종 부동소수점(Floating Point)으로 형 변환하여 출력함에 있어서 발생하는 형 변환 상의 반올림 오차 수준임을 확인하였다.

3. 레이더 신호 후처리 로직 구현

Fig. 9는 System Generator를 이용해 구현한 후처리 로직 블록도이다. 후처리 과정 역시 운용 파형에 따라 두 종류의 독립적인 신호처리 경로를 거치지만, 로직 공용화를 위해 외부설정으로 분기하여 동작하도록 설계하였다. 로직은 신호처리 개념에 의해 채널보정, FFT, NCI, CFAR 등의 기능블록을 포함한다.

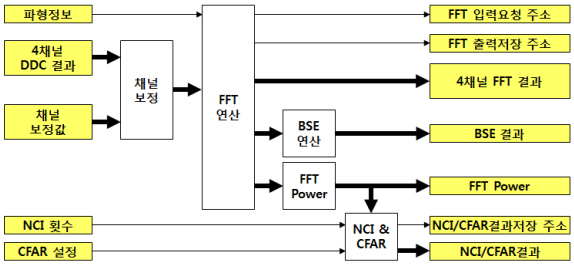

Fig. 9. Postprocessing logic

연산이 시작되면 블록에서는 입력된 파형정보에 따라 FFT 계산에 필요한 데이터를 순차적으로 요청하고,

블록 외부에서는 요청 데이터에 맞는 DDC 결과를 입력하여 FFT 연산이 진행된다. FFT 연산이 시작 되고 얼마간의 지연(Latency) 후 FFT 결과가 출력되기 시작하며, 동시에 해당 데이터가 저장될 메모리 주소도 함께 출력된다. FFT 계산결과는 “BSE 연산”과 “FFT Power” 블록을 거치고, “FFT Power” 결과는 다시 “NCI & CFAR” 블록에 입력된다.

3.1 채널보정

Fig. 9에서 “채널보정”으로 명시된 블록은 수신기 및 케이블 등 하드웨어 지연 및 경로손실에 의해 발생되는 채널 간 이득 및 위상차를 디지털적으로 보정하기 위한 블록이며, 주 채널 기준으로 나머지 부 채널의 이득과 위상을 보정한다. 보정값은 외부 프로세서에서 계산한 위상차를 FPGA로 입력할 수 있게 설계하였으며, 채널보정 과정은 각 채널 신호 데이터와 외부에서 받은 복소수 보정값의 곱으로 계산된다.

3.2 FFT(Fast Fourier Transform) 연산

Fig. 10은 Fig. 9에서 “FFT 연산”로 명시된 블록의 내부이며, FFT 연산에 필요한 데이터 입출력 타이밍 제어, 제로패딩(Zero-Padding) 및 윈도윙(Windowing) 등의 기능을 포함한다. 그림에서 MCODE로 명시된 블록은 System Generator의 MCode 블록으로 구현한 기능이다. MCode 블록은 제한된 기능의 MATLAB 언어를 사용해 간단한 고정소수점(Fixed Point) 연산을 할 수 있다. MCode 블록 연산은 기준 클럭 당 한번 수행되며 “Persistent State”로 선언된 변수를 사용하여 내부적으로 변수 값 및 상태를 유지할 수 있다[4].

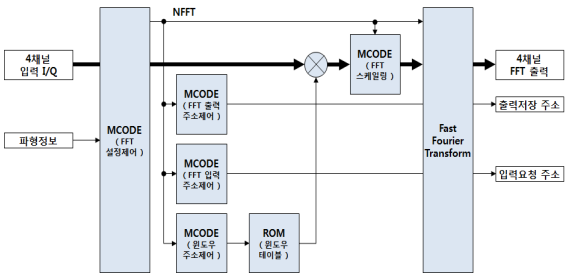

Fig. 10. FFT calculation block

그림에서 “FFT 설정제어” 블록은 운용 파형에 따라 FFT 데이터 개수, 스케일링 등의 FFT 블록 설정에 필요한 파라미터를 동적으로 생성한다. “FFT 입력주소 제어” 기능은 FFT 입력 데이터 요청을 위해, 전처리 결과가 저장되어있는 메모리의 읽기주소를 생성하고, “FFT 출력주소 제어”는 FFT 결과 저장을 위한 메모리 쓰기주소를 생성한다. “윈도우 테이블”에는 운용 파형별 윈도우 계수가 저장되어 있으며, “윈도우 주소 제어” 블록은 운용 파형 및 FFT 입력 데이터 시퀀스에 따라 해당 데이터에 해당되는 윈도우 값을 취하기 위해 윈도우 테이블(ROM)의 읽기주소를 생성한다.

3.3 BSE(Boresight Error) 연산

BSE는 안테나 LOS(Line of Sight) 기준, 표적의 시선각으로 합·차 채널의 FFT 결과를 이용해 계산된다. 일반적인 모노펄스의 각도 측정값은 신호에 대한 합·차 채널의 복소수 전압비로 표현되며 식 (2)와 같다[5].

$$v_{error}(\theta) = \frac{\Sigma \bullet \Delta}{|\Sigma|^2} \tag{2}$$

여기서 Σ와 Δ는 합·차 채널의 복소수 측정값이다.

Fig. 11은 BSE 연산 블록이다. FPGA가 나누기 연산에 취약하기 때문에 나누기 연산 제거 및 로직 단순화 목적으로 합 채널의 전력 $|\Sigma|^2$ 만큼 가중된 값을 계산하도록 하였다.

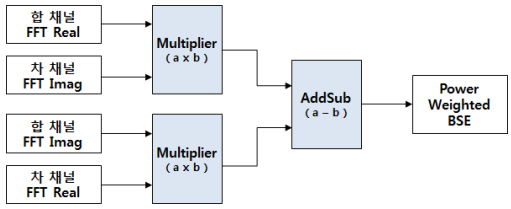

Fig. 11. BSE calculation block

3.4 FFT Power

일반적으로 레이더에서는 표적 탐지를 위해 선형 검파기(Linear Detector) 또는 제곱 검파기(Square Law Detector)를 사용하는데, 두 검파기는 신호 상태나 주변 환경에 따라 CA-CFAR(Cell Average Constant False Alarm Rate)의 표적 탐지 및 오 탐지 확률의 성능이 달라진다[6].

본 연구에서는 제곱 검파기를 사용하였으며 FFT Power 블록은 표적 탐지에 사용되는 제곱 검파 값인 FFT 스펙트럼의 전력을 계산한다. Fig. 12는 FFT 스펙트럼 전력 계산 블록이다.

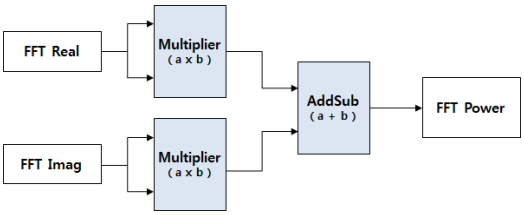

Fig. 12. FFT power calculation block

3.5 NCI(Non-Coherent Integration)

NCI는 신호의 크기정보를 누적하는 방법으로 여러 CPI(Coherent Processing Interval) 동안 획득한 스펙트럼 전력을 누적하여 잡음의 분산을 낮추고 작은 표적 신호에 대한 탐지확률을 높이는 효과가 있다[6].

Fig. 13은 NCI와 CFAR 연산 블록을 나타낸다. NCI를 수행하기 위해서는 이전 CPI에서 계산된 스펙트럼 누적 결과에 현재 CPI에서 계산된 스펙트럼 결과를 더하여 저장해야하기 때문에 최소 2개의 메모리가 필요한데, 그림에서 BRAM(Block RAM)이 NCI 결과를 저장하는 2개의 메모리 블록이다. MCode로 구현된 "NCI 제어" 블록은 2개의 BRAM을 파형 및 누적 횟수에 따라 관리하며, 메모리 초기화, 입력 데이터 동기에 맞는 NCI 메모리 읽기·쓰기 주소 생성, 메모리 절체 등의 기능을 수행한다. 계산된 NCI 결과는 다음 절에서 언급할 CFAR 블록의 입력이 된다.

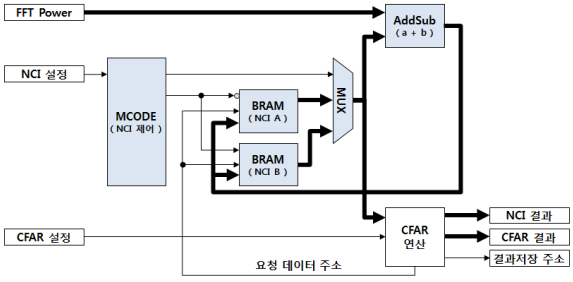

Fig. 13. NCI & CFAR calculation block

3.6 CFAR(Constant False Alarm Rate)

3.6.1 CFAR 이론

CFAR 검출기는 레이더에서 일반적으로 사용하는 탐지 기법으로, 측정치에서 잡음, 클러터 등 간섭신호에 대한 크기를 확률적으로 예측하고, 간섭신호에 의한 오표적 검출 확률을 특정 값 이하로 유지할 수 있도록 임계값을 설정한다[7].

Fig. 14는 1차원 CFAR의 처리개념이다. 데이터 윈도우에서 CFAR 윈도우가 이동(Shift) 하면서 CFAR 테스트가 수행된다. 테스트 셀(CUT : Cell Under Test) 중심으로 가드 셀(G: Guard)을 제외하고 전·후의 기준(Lagging and Leading Reference) 셀을 취한다. 이 기준 셀을 이용, 확률적 방법에 의해 잡음이나 클러터 등의 간섭 신호 크기 $\hat{g}$를 예측하고, 오표적 검출확률이 특정 값 이하가 되도록, CFAR 상수 α를 곱하여 CFAR 임계값을 계산한다[7]. 테스트 셀의 값이 계산된 CFAR 임계값보다 크면 표적 후보로 선택된다.

본 연구에서 구현한 CFAR는 CA(Cell Average)와 OS(Order Statistics) 방식으로 CA는 기준 셀들의 평균값을, OS는 기준 셀들을 크기 순서로 정렬 한 후 지정된 선택 비율에 해당되는 셀 값을 선정한다.

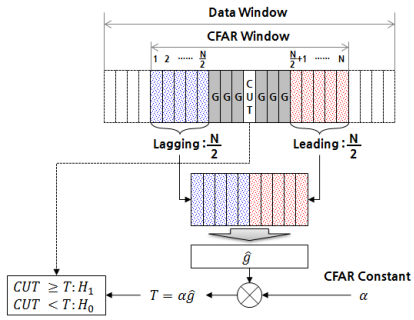

Fig. 14. Processing concept of 1D CFAR

Fig. 15는 2차원 CFAR의 처리개념이다. 거리·도플러의 2차원 맵으로 구성된 데이터 윈도우에서 2차원 CFAR 윈도우가 거리·도플러 방향으로 이동하면서 CFAR 테스트를 수행한다. 테스트 셀(CUT: Cell Under Test) 중심으로 주변에 가드 셀을 두고 2차원으로 기준 셀을 취한다. 기준 셀을 취하는 방식은 간섭 신호의 특성에 따라 테스트 셀의 주변 혹은 상하좌우 셀을 취하게 된다.

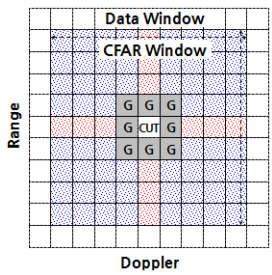

Fig. 15. Processing concept of 2D CFAR

3.6.2 CFAR 구현 – 개요

Fig. 16은 Fig. 13의 “CFAR 연산” 블록 내부이다. MCode로 구현한 “CFAR 제어” 블록은 CFAR 연산 시퀀스에 맞추어 입력 데이터(NCI 결과)가 저장된 메모리를 엑세스한다. “1D·2D CFAR 쉬프트 메모리” 블록은 Fig. 14, 15의 개념에 따라 CFAR 윈도우 영역에서 테스트 셀과 기준 셀을 출력한다. 기준 셀은 “1D·2D CFAR 임계값 계산” 블록에 입력되어 CA 및 OS CFAR의 임계값 계산에 사용된다.

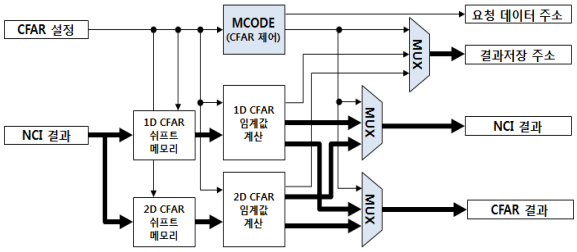

Fig. 16. CFAR calculation block

3.6.3 CFAR 구현 – 쉬프트 메모리

Fig. 17은 Fig. 16에서 “1D CFAR 쉬프트 메모리”로 명시된 블록의 내부이다. 블록은 CFAR 윈도우를 형성하는 레지스터 열로 구성되어 있으며 테스트 셀과 기준 셀을 출력한다. “1D CFAR 출력 관리”는 연산 시퀀스에 맞게 CFAR 임계값 및 NCI 결과(테스트 셀) 값이 저장될 메모리 쓰기주소를 생성하며, CFAR 윈도우 셀의 유효성을 판단한다.

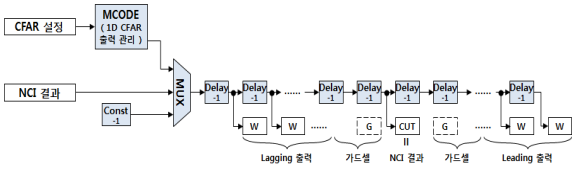

Fig. 17. Shift memory block for 1D CFAR

1D CFAR를 사용하는 파형은 협대역 통과 필터를 사용하는, 경로 1의 전처리 과정을 거친다. 따라서 CFAR 계산 시 협대역 필터 통과영역 내의 데이터만 사용해야하며, 따라서 데이터 윈도우는 필터 통과영역의 데이터 영역이 된다. 3.6.1항에서 언급한 바와 같이 데이터 윈도우에서 CFAR 윈도우가 이동하면서 CFAR 계산을 수행하는데, 만약 CFAR 윈도우가 데이터 윈도우(협대역 필터 통과영역)의 경계에 위치하여 CFAR 윈도우 내의 일부 데이터가 유효하지 않을 경우가 고려되어야 한다. “1D CFAR 출력 관리” 블록은 CFAR 윈도우 셀의 유효성을 판단하여, 이후 블록에서 유효하지 않은 셀을 CFAR 임계값 계산에 사용하지 않도록 MUX를 이용해 정해진 무효(invalid) 값을 입력한다.

“2D CFAR 쉬프트 메모리” 구성 역시 기본 개념은 Fig. 14의 2차원 CFAR 처리개념을 따른다. 그러나 한 방향으로 이동하는 1D CFAR 쉬프트 메모리와는 달리 CFAR 윈도우가 데이터 윈도우 영역을 거리·도플러 양방향으로 이동하므로 구성이 복잡해진다. 2D CFAR의 경우 CFAR 윈도우 영역이 2차원으로 구성되기 때문에 순차적으로 데이터를 입력하는 경우 CFAR 윈도우 영역이 틀어지는 현상이 발생하므로, 테스트 셀이 이동하더라도 윈도우 영역이 틀어지지 않도록 데이터가 입력되어야 한다. 또한 CFAR 윈도우가 데이터 윈도우의 경계에 있을 때 CFAR 윈도우가 거리·속도 모호성에 의해 폴딩(Folding)되는 경우도 고려해야 한다.

3.6.4 CFAR 구현 – 임계값 계산

Fig. 18은 Fig. 16에서 “1D/2D CFAR 임계값 계산”으로 명시된 블록의 내부이다. 1D와 2D 블록은 기준 셀 개수만 다를 뿐 동일한 구조를 갖는다. 블록 입력은 앞서 언급한 “쉬프트 메모리 블록”에서 출력된 기준 셀과 CFAR 종류선택 및 CFAR 임계값 연산에 필요한 OS CFAR 선택 비율, CFAR 상수 등의 파라미터이다. “CA-평균계산” 블록은 유효한 기준 셀들의 평균을 계산한다. “OS-정렬” 블록은 유효한 기준 셀을 크기순으로 정렬하고 선택비율에 해당하는 값을 출력한다. 임계값 계산 출력은 CA 또는 OS에서 계산된 출력 값에 CFAR 상수를 곱한 값이 출력된다.

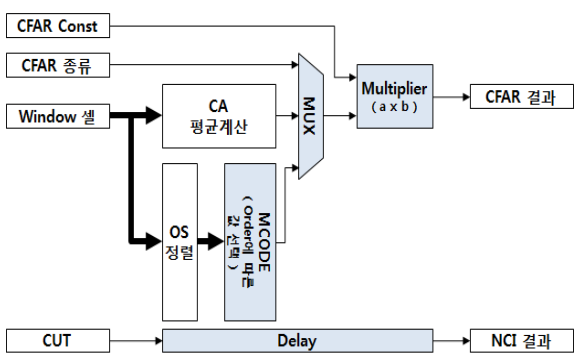

Fig. 18. Threshold calculation of CFAR

Fig. 19는 Fig. 18에서 “CA 평균계산”으로 명시된 블록의 내부이다. 입력된 기준 셀 값에 대하여 유효성을 판단하여 데이터 값과 유효성(유효 시 1)을 출력하고, 단계별로 합산된 유효 셀 값을 유효 셀 개수로 나누어 평균값을 출력한다.

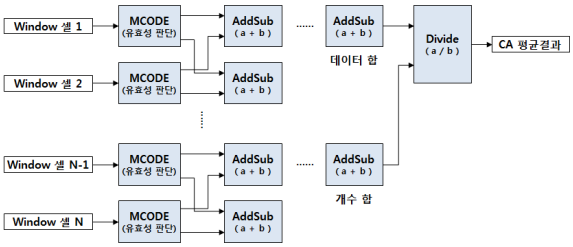

Fig. 19. Average calculation of CA-CFAR

Fig. 20은 Fig. 18에서 “OS 정렬”로 명시된 블록의 내부이다. 정렬 알고리즘은 병렬처리 시 효율적인 방법으로 알려진 Batcher’s even-odd 알고리즘을 사용하였다[8]. 블록은 N개의 기준 셀을 입력받아 N/2 정렬을 먼저 수행하고, 그 결과를 이용하여 나머지 정렬과정을 수행한다. N/2 정렬은 동일하게 N/4 정렬, N/8 정렬 등의 하위 블록을 포함한다. 정렬시 기본적으로 사용되는 “비교 및 교환”은 MCode로 구현하였다.

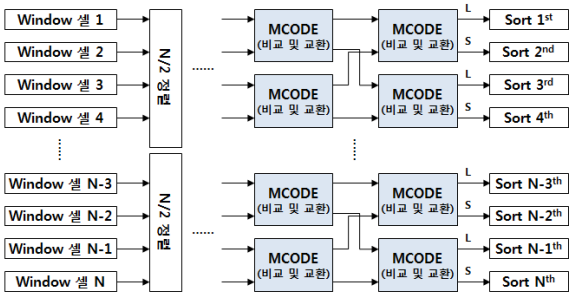

Fig. 20. Sorting for OS-CFAR

3.6.5 2D OS-CFAR의 연산속도 비교

본 연구에서 가장 괄목할 연구결과는 2D OS-CFAR를 FPGA로 구현한 것이다. 2D OS-CFAR에서는 기준 셀을 이차원으로 취하는데, 테스트 셀이 이동할 때마다 달라지는 기준 셀을 매번 메모리에서 액세스할 경우, FPGA에서는 메모리 액세스 시 1 클럭이 소모되므로 연산시간은 데이터 윈도우의 셀 개수와 CFAR 연산에 사용되는 기준 셀 개수의 곱에 비례하여 증가하게 된다. 하지만 본 연구에서는 테스트 셀이 이동할 때마다 취해지는 기준 셀을 쉬프트 메모리를 이용한 Pile-Line 방식으로 구현했기 때문에 연산시간은 데이터 윈도우의 셀 개수에만 비례하게 된다.

본 연구에서 구현한 FPGA 2D OS-CFAR의 연산속도를 기존에 FPGA로 구현한 결과와 비교하고자 하였으나 국내·외에 유사한 연구 사례 및 문헌이 없기 때문에, 연산성능을 간접적으로 확인하기 위하여 CUDA (Compute Unified Device Architecture) GPU(Graphic Processing Unit)를 이용한 2D OS-CFAR 구현 결과[9]를 사용하였다. 참고논문에서는 2D OS-CFAR는 400×300 데이터 윈도우 20세트에, 기준 셀 개수와 K Value 변화 조건에 따른 연산시간을 측정하였다. Table 1은 참고논문의 GPU기반 CUDA runtime 라이브러리를 사용하여 측정한 연산시간(참고문헌 [9]의 Figure 8 그래프를 대략적 수치로 변환)과 본 논문에서 구현한 FPGA 로직의 연산시간을 비교한 표이다. FPGA 동작 클럭은 100 MHz를 사용하였다.

Table 1. Comparison of Processing Time for 2D OS-CFAR

연산조건		연산시간 [ms]	
기준 셀 개수	K value	FPGA	GPU
8	6	0.0240	0.120
12	9	0.0240	0.120
16	12	0.0241	0.130
20	15	0.0241	0.150
24	18	0.0241	0.160

본 논문에서 구현한 FPGA 기반의 2D OS-CFAR가 참고문헌 대비 작게는 5배, 크게는 6배 이상의 연산속도 향상이 있음을 확인할 수 있었으며, 윈도우 개수가 증가할수록 연산시간의 차이가 더 커짐을 알 수 있다. GPU의 병렬처리 기법을 이용하면, 1대 1 비교 및 교환에 필요한 평균 연산횟수인 $n \times \log(n)$[10]에 소모되는 연산시간을 단축할 수는 있지만, 윈도우 이동 시 새로운 기준 셀 값을 정렬함수에 입력하기 위해 발생하는 메모리 할당 및 데이터 복사 등에 소모되는 시간은 해결할 수 없다.

본 논문에서 구현한 OS-CFAR의 경우에는 앞에서 언급한 바와 같이, 쉬프트 메모리를 이용한 Pile-Line

방식으로 구현했기 때문에, 전체 연산시간은 연산하고자 하는 총 윈도우 셀 개수에만 비례하며, 정렬하고자 하는 기준 셀 개수에는 거의 영향을 받지 않는다. 예를 들어 16개 데이터와 32개 데이터 정렬에 필요한 연산시간은 각각 17클럭, 21클럭인데, 이는 총 윈도우 셀 개수×4클럭(21-17)의 연산시간 차이가 아니라 단지 4클럭의 연산지연(Latency)으로 나타나기 때문이다.

4. 레이더 신호 후처리 로직 검증

후처리 로직 검증은 앞의 전처리 로직과 동일하게 MATLAB 코드로 계산한 결과와 비교하였다. Fig. 21은 신호처리 경로 2에 대한 CPI 후처리 결과이며, Fig. 22, 23은 Fig. 21의 CPI 후처리 결과를 NCI(파란색)하고 2D OS-CFAR 임계값(초록색)을 계산한 결과이다. 그림에서 임계값을 통과한 셀은 빨간 원으로 표시하였다. 분석결과 FPGA로 구현한 신호처리 기능들이 정상 동작함을 확인하였다.

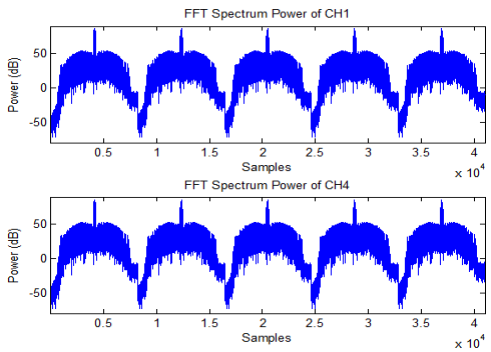

Fig. 21. Signal processing result of case 2

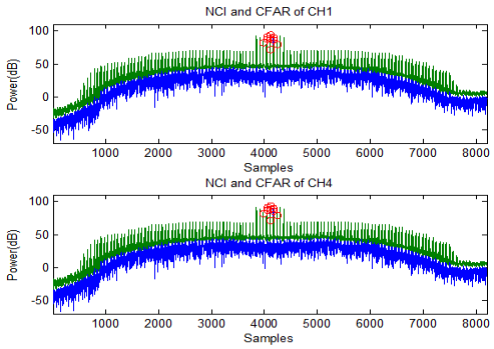

Fig. 22. NCI and 2D OS-CFAR results of case 2

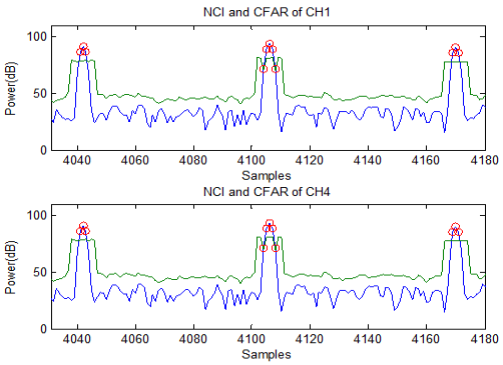

Fig. 23. Zoom in of Fig. 22

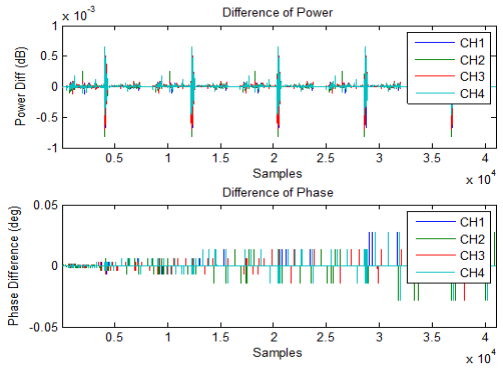

Fig. 24. FFT result comparison : spectrum power & phase differences

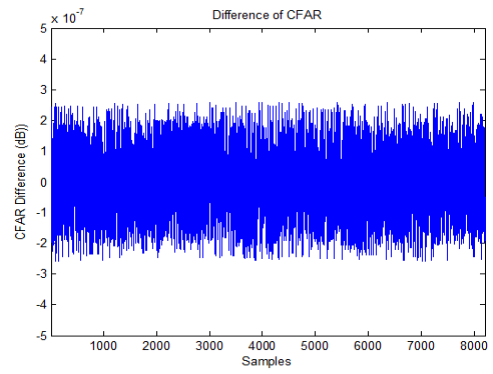

Fig. 25. OS-CFAR result comparison

Fig. 24는 FPGA에서 각 채널 별로 FFT한 결과를 MATLAB 계산결과와 비교한 그림이며, 비교 결과 전력은 최대 0.85E-3dB 미만, 위상은 최대 0.03° 미만의 차이를 보였다. Fig. 25는 FPGA 채널 1의 CFAR 결과

를 MATLAB 계산결과와 비교한 그림이며, 비교 결과 CFAR 임계값은 최대 3.0E-7dB 미만의 차이를 보였다. 여기서 특이 사항은 Fig. 24의 FFT 스펙트럼 전력비교에서 표적 신호 부근의 오차가 두드러짐을 확인할 수 있는데, 이는 FPGA에서 구현한 부동소수점 연산은 32비트 Float 형을 사용하였고, MATLAB에서 계산한 비교 값은 64비트 Double 형을 사용하였기 때문에, 비교적 큰 값을 갖는 표적신호 부근에서의 오차가 두드러지게 나타나 보였다.

5. 결론 및 향후계획

본 논문에서는 FPGA를 이용한 레이더 신호처리 구현을 위해 System Generator 기반의 신호처리 로직 설계에 대해 기술하였고, 개발된 로직의 기능 및 성능을 검증하기 위해 MATLAB 연산 결과와 비교를 수행하였다.

Table 2는 본 연구에서 구현한 FPGA 신호처리 로직을 System Generator의 Resource Estimation 기능을 이용해 산출한 FPGA 자원 소모 개수이다. 자원 소모 수준으로 보았을 때, 최근 FPGA와 Multi-Core ARM 프로세서가 One Chip SoC 형태로 출시되고 있는 플랫폼, 예를 들어 Xilinx사의 ZYNQ나 Altera의 Arria 10 등을 이용한 소형 신호처리기 개발이 가능할 것으로 판단된다.

Table 2. Resource estimation of each function

구분	BRAMs	DPSs	LUTs	Registers
전처리 (경로 1)	10.5	72	5813	6501
전처리 (경로 2)	2	416	22388	32174
후처리	87.5	347	150958	109831
합계	**100**	**835**	**179159**	**148506**

본 논문에서 구현한 FPGA 로직은 현재 실제 하드웨어에 탑재하여 시험 중이며, 그 외에 연산부하가 많은 것으로 알려진 클러스터링 알고리즘과 STAP(Space-Time Adaptive Processing) 알고리즘을 FPGA로 구현 중에 있다.

References

[1] “GPU vs FPGA Performance Comparison,” White Paper, BWP001 v1.0, BERTEN.

[2] H. A. Said, A. A. El-Kouny and A. E. El-Henawey, “Design and Realization of Digital Pulse Compression in Pulsed Radars Based on Linear Frequency Modulation(LFM) Waveforms Using FPGA,” International Conference on Advanced Information and Communication Technology for Education, pp. 827-832, 2013.

[3] Enrique Escamilla-Hernández, Victor Kravchenko, Volodymyr Ponomaryov, Dniel Robles-Camarillo and Luis E. Ramos V., “Real Time Signal Compression in Radar Using FPGA,” Cientifica, Vol. 12, No. 3, pp. 131-138, 2008.

[4] XILINX Inc, “System Generator for DSP, Reference Guide, UG638(v14.5),” pp. 283-304, 2013.

[5] William L. Melvin and James A. Scheer, “Principles of Modern Radar, Advanced Techniques,” Scitech, pp. 559-560, 2013.

[6] Mark A. Richards, James A. Scheer and William L. Holm, “Principles of Modern Radar, Basic Principles,” Scitech, pp. 560-587, 2010.

[7] Mark A. Richards, James A. Scheer and William L. Holm, “Principles of Modern Radar, Basic Principles,” Scitech, pp. 589-623, 2010.

[8] Slobodan SIMIĆ, Milenko ANDRIĆ and Bojan ZRNIĆ, “An FPGA Based Implementation of a CFAR Processor Applied to a Pulse-Compression Radar System,” Radioengineering, Vol. 23, No. 1, pp. 73-83, April 2014.

[9] Hans Erik Fjeld, “Application of Parallel Programming in a Automatic Detector for a Pulsed MTD Radar system,” Norwegian University of Science and Technology Department of Electronics and Telecommunications, June 2012.

[10] “Sorting Algorithm”, WIKIPEDIA, https://en.wikipedia.org/wiki/Sorting_algorithm