cadence®

Fixed- and Floating-Point FMCW Radar Signal Processing with Tensilica DSPs

Automotive advanced driver assistance system (ADAS) applications increasingly demand radar modules with better capability and performance. These applications require sophisticated radar processing algorithms and powerful digital signal processors (DSPs) to run them. Because these embedded systems have limited power and cost budgets, the DSP's instruction set architecture (ISA) needs to be efficient and easy to program. This paper describes how to optimize typical frequency modulated continuous wave (FMCW) radar range-Doppler signal processing algorithm kernels using both fixed- and floating-point computations on Cadence® Tensilica® DSPs.

Contents

Introduction

Radar technology is playing an increasingly important role in automotive ADAS applications. These applications require higher performance and more capabilities from the radar module to determine distance, direction, and speed of targets in a multi-target scenario.

Because radar modules typically operate in multiple modes—for example, a "search mode" to scan for objects and "track mode" to track specific objects of interest—adaptability and multi-function operation are essential. Using a programmable DSP as the radar signal processor offers many advantages. Implementing radar signal processor functions as software modules enables functionality to be easily adapted by changing the code. However, the DSP executing these algorithms must have an efficient and well-tuned instruction set architecture (ISA) to enable sophisticated processing algorithms to extract information from the radar signals.

This paper describes how to optimize algorithm kernels for FMCW radar range-Doppler processing on the following Cadence® Tensilica® DSPs:

- Tensilica Fusion G3 DSP—A very long instruction word (VLIW) vector SIMD architecture targeting multi-purpose applications with an optional floating-point unit[1]
- Tensilica ConnX BBE32EP—A 16-way VLIW SIMD DSP, supporting 16-bit and 32-bit fixed-point complex vector operations[2]

Because system parameters determine the computational complexity of the algorithms, this paper presents kernel performance in the context of a set of typical system parameters.

Optimizing signal processing algorithms for a DSP requires careful consideration of a variety of factors. One factor is the data type and precision to be used for implementing computations for each of the algorithm kernels. The

algorithm kernel should be precise enough to meet the functional requirements of the target application, based on the analog-to-digital converter (ADC) resolution, the amount of noise in the RF and analog front-end circuits, and the application's target detection requirements.

Designs should minimize the local data memory requirement while ensuring that processing kernels operate on data from local memory for maximum processing efficiency. Finally, core computations should be performed efficiently, minimizing compute cycles and memory accesses.

This paper describes the FMCW algorithm kernel analysis and optimizations, including digital beamforming (DBF), 2D range-Doppler fast Fourier transforms (FFTs) with windowing, and 2D constant false alarm rate (CFAR) kernels.

FMCW Processing Chain

The computational complexity of the algorithms depends on system parameters. This paper presents algorithm kernel performance in the context of the representative system shown in Figure 1, using a linear sawtooth FMCW waveform.

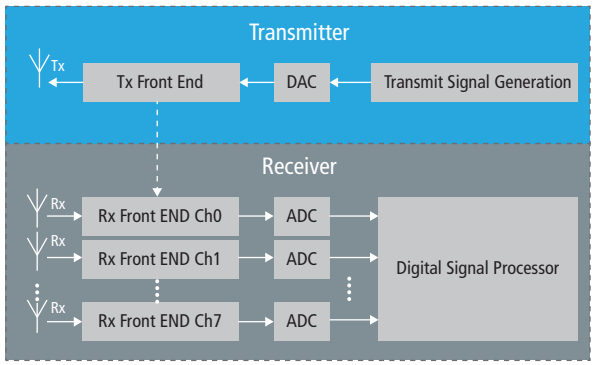

Figure 1: Radar System Block Diagram

This paper presumes that the receiver has an antenna array with eight elements for receive DBF. Each antenna array element generates one of the baseband signal's receive channels. The transmitter is shown as a single antenna element in Figure 1, but can also be an antenna array with transmit beamforming to focus transmitted energy in beams along specific directions. This paper supposes the transmitted beam illuminates the desired area of interest, and we perform signal processing on the receiver end.

In linear sawtooth FMCW, the transmitted waveform is a sequence of linear chirp signals, and the reflected received signals are mixed with the transmitted signals to generate the beat signal. The beat signal is the baseband signal, which is sampled by the ADCs and sent to the DSP. The duration of one chirp is denoted by $\boldsymbol{T_f}$, and the total duration of the multiple chirp sequence transmission is called the coherent processing interval (CPI), or T_{CPI}. For one CPI, the ADCs generate a data sequence that is sampled along three dimensions: *range dimension* (time sampling of one chirp), *Doppler dimension* (multiple chirps), and *channel dimension* (each antenna element). This three-dimensional data sequence for one CPI is the received radar data cube, which is processed by the signal processing algorithm chain on the DSP. The sizes of the various dimensions are chosen based on required range, speed, and direction resolutions. This white paper uses the following typical system parameters:

- λ = 3.9mm, corresponding to 77GHz radar
- $\boldsymbol{B}$ = 150MHz, the bandwidth sweep of one chirp
- $\boldsymbol{T_f}$ = 10µs
- Range maximum = 150m, range resolution = 1m
- Relative speed maximum = 100km/hour
- Beat signal sampling frequency = 30Msps
- Data cube range dimension size = 512
- Data cube Doppler dimension size = 64
- Data cube channels dimension size (number of antenna array elements) = 8
- T_{CPI} interval = 25ms

Implementing the Signal Processing Chain

The signal processing chain is implemented using multiple algorithm kernels that are optimized using the DSP ISA, and stitched together to form a complete processing chain using appropriate data buffers and data flow. The data memory scheme is as follows:

- An on-chip system memory is used to store larger buffers.
- Single-cycle local data memories are used for storing smaller working buffers and input/output buffers of the algorithm kernels.
- The optimized algorithm kernels use the local data memory buffers for input/output.

Each kernel processes data by blocks. A double buffering scheme is used to enable data transfers between the system and local data memories in parallel to kernel processing as shown in Figure 2.

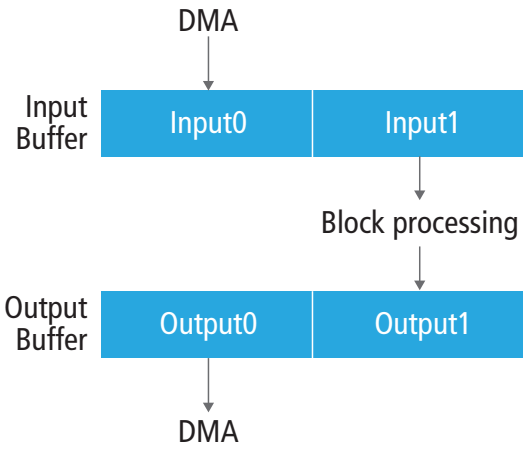

Figure 2: Double Buffering

The data block size and the tiling structure are chosen to minimize data transfers and ensure there are fewer transfer cycles than compute cycles.

The input data cube (range = 512 samples, Doppler = 64, and channels = 8) is generated using the system parameters listed in the FMCW Processing Chain section for scenarios with simulated targets. The output of the processing chain is verified for simulated targets.

Fusion G3 DSP Features and Configuration

The high-performance Fusion G3 DSP supports floating-point and fixed-point DSP operations with a comprehensive range of data types, in addition to the scalar Xtensa® real-time controller, making it ideal for a wide mix of compute-intensive signal processing and control applications. This DSP supports operations on 8-, 16-, 32-, and 64-bit signed and unsigned integer types, and single- and double-precision floating-point types. It supports multi-way vectorization, depending upon the data type (integer/float/double-precision), as well as 2/4/8/16-way multiplier-accumulators (MACs) with 20/40/80-bit accumulation precision.

This example uses single-precision floating-point computations for all kernels on the Fusion G3 DSP. For simulation and cycle measurements, the following Fusion G3 DSP configuration is used:

- Two 64KB local data RAMs: internal data memories with two banks each
- One 256KB local instruction RAM: internal instruction memory
- 8MB SRAM: system memory that can be accessed through a system bus

ConnX BBE32EP DSP Features and Configuration

The ConnX BBE32EP is an ultra-high-performance DSP architecture designed for next-generation complex signal processing applications. It combines 16-way single instruction, multiple data (SIMD), 32 16x16-bit MACs, and up to a 5-issue VLIW architecture with a set of versatile pipelined execution units. These units support flexible-precision real and complex multiply-add, bit manipulation, data shift and normalization, data select, shuffle, and interleave operations.

This example uses fixed-point computations for all kernels on the ConnX BBE32EP DSP. For simulation and cycle measurements, the following ConnX BBE32EP DSP configuration is used:

- Two 128KB local data RAMs: internal data memories with two banks each
- One 128KB local instruction RAM: internal instruction memory
- 4MB system RAM: system memory that can be accessed through a system bus
- FFT ISA option

Algorithm Kernels

The signal processing algorithm chain for processing one data cube is shown in Figure 3.

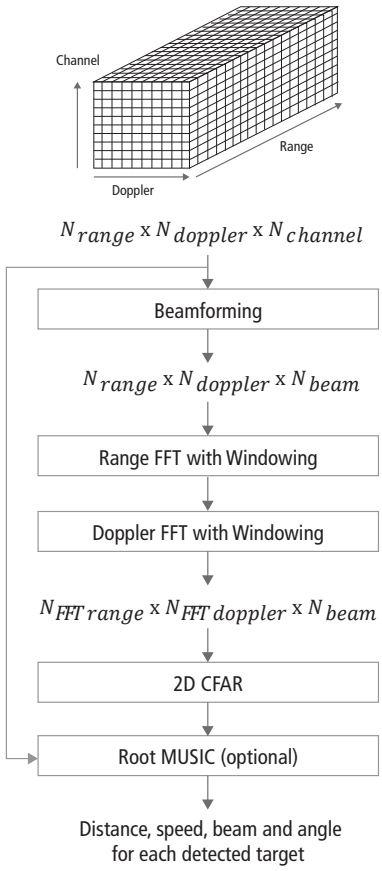

Figure 3: Signal Processing Algorithm Chain to Process a Data Cube for one CPI

Digital Beamforming (DBF) Kernel

The DBF kernel performs FIR filtering along the channel dimension in the data cube. The length of the FIR filter is equal to the number of antenna elements (eight in this case), and the FIR taps are pre-chosen to pass signals from a beam centered on a particular pre-determined spatial direction, and to suppress signals from other directions. Thus, the DBF takes the data cube as input, and produces a range-Doppler 2D signal as an output beam. Different sets of FIR taps can be used to generate multiple beam signals using the same data cube. In this paper, five beams are generated along different directions using separate sets of FIR taps. This kernel can be implemented in Tensilica DSPs using the information shown in Table 1.

Description	Fusion G3 DSP (Floating-Point)	ConnX BBE32EP DSP (Fixed-Point)
Input data cube size	512x64x8	
Output data cube size for five beams	512x64x5	
Input sequence	256 complex (floating-point) samples	512 complex 16-bit (fixed-point) samples (represented in Q15 format)
Optimization	Vectorize the core loop using vector load, multiply accumulate, and vector store instructions	Vectorize the core loop using the vector load, complex multiply accumulate, and vector store instructions

Table 1: DBF Kernel Processing

2D Range-Doppler FFT Kernel

The 2D range-Doppler FFT kernel is used to transform each of the five 2D beam signals produced by the DBF kernel to a 2D frequency domain spectrum. A windowed 2D FFT is used to separate the signals, with the range FFTs along the range dimension, followed by Doppler FFTs along the Doppler dimension. Each 1D FFT also uses a hamming window.

The range FFT is computed as a 1D FFT, whereas the Doppler FFT is performed as a block-based FFT to avoid transposing the input and output. The FFT algorithm is based on the Kronecker product-based formalization.[3] Note that the range and Doppler FFT kernels use different block sizes. In the last stage, the Doppler FFT kernel computes a point-wise energy of each bin. The range-Doppler FFT kernel can be implemented in Tensilica DSPs using the information shown in Table 2.

Description	Fusion G3 DSP (Floating-Point)	ConnX BBE32EP DSP (Fixed-Point)
Range FFT kernel—input processing block size	Two rows of complex floating-point samples	Four rows of complex 16-bit (fixed-point) samples
Doppler FFT kernel—input processing block size	16 columns of complex floating-point samples	32 columns of complex 16-bit (fixed-point) samples Note: Doppler FFT kernel performs a vectorized block transpose using interleave/de-interleave instructions.
Optimization	Vector floating-point instructions such as load, store, combined add and subtract for FFT, conjugate, and MACs	The radix-4 and radix-2 DIF butterflies are vectorized using FFT instructions with 16-bit precision and dynamic scaling after each stage

Table 2: 2D Range-Doppler FFT Kernel Processing

Constant False Alarm Rate (CFAR) Kernel

The CFAR kernel operates on this 2D energy signal in the frequency domain, classifying each bin or cell under test (CUT) in the 2D FFT energy signal either as a target bin or noise/clutter bin by comparing each CUT with a scaled estimate of noise + clutter. The scale factor and threshold used is determined by the probability of false alarm (thus the name CFAR). Noise + clutter is estimated by considering the cells in a rectangular neighboring region for a given CUT, leaving out some immediate neighbors of the CUT (called the guard region) due to potential energy leakage from the CUT into its immediate neighbors. This implementation uses a 5x9 cell area centered at the CUT as the training region, and a 3x5 area centered at the CUT as the guard region. See Figure 4.

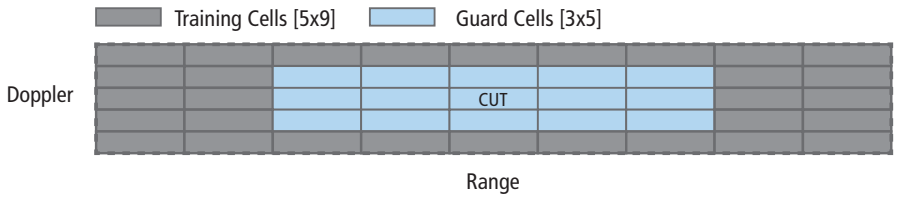

Figure 4: CFAR Processing Window

If the CUT value is greater than the weighted noise + clutter estimate, then the CUT is classified as a target, otherwise the CUT is classified as noise/clutter. The CFAR kernels are distinguished by how the noise estimate is formed using the training cells. For example, the cell-averaging constant false alarm rate (CA-CFAR) noise estimate is formed by taking an average of the cells in the training area, whereas the ordered statistic constant false alarm rate (OS-CFAR) noise estimate is formed by sorting the training area's cell values in descending order, and using the Nth sorted value. Various combinations of CA-CFAR and OS-CFAR can also be used. The AND CFAR combination does a logical AND of the classification outcomes of the CA-CFAR and OS-CFAR for each CUT, while the OR CFAR combination does a logical OR. In this example, AND CFAR is used. The CFAR kernel can be implemented in Tensilica DSPs using the information shown in Table 3.

Description	Fusion G3 DSP (Floating-Point)	ConnX BBE32EP DSP (Fixed-Point)
Input data processing block size for five beams	8x32	8x64
Access block size (to include all neighborhood samples for all CUTs)	12x40	12x72
CA-CFAR kernel optimization	Averages four CUTs at a time	Uses a select instruction to discard the guard cell elements and only select the elements needed for cell averaging
OS-CFAR kernel optimization	Requires the N^{th} maximum from the buffer of training cells Uses a two-step process: 1. Data in training cells for four CUTs is arranged in a streaming order 2. On this data, the Nth maxima is computed for four CUTs at a time	Requires finding the N^{th} maximum from the buffer of neighboring cells Note that this kernel is optimized using the max category of operations. (Refer to the *ConnX BBE32EP User's Guide*, Table on *Sample Categories of Operations*.)

Table 3: CFAR Kernel Processing

Performance Measures

For a test vector with data cube dimension of 512x64x8 with a coherent processing interval (CPI) repetition time of 25msec, the floating-point implementation on the Fusion G3 takes less than 240MHz, and the fixed-point implementation on the ConnX BBE32EP takes less than 140MHz. For detailed results on these computations, refer to the *Implementing FMCW Radar Signal Processing Algorithms on the Tensilica Fusion G3 DSP and Implementing FMCW Radar Signal Processing Algorithms on the Tensilica ConnX BBE32EP DSP* application notes.

Note that the Fusion G3 DSP can also be used for high-resolution angle estimation using the multiple-signal classification (MUSIC) algorithm, where floating point is required for low-power real-time operation.

Summary

This paper presents an optimized implementation of typical FMCW radar range-Doppler signal processing algorithm kernels on the Fusion G3 and ConnX BBE32EP DSPs. Performance results show that the Cadence Tensilica DSPs with fixed- or floating-point computations are well-suited for today's embedded radar processing algorithms.

Additional Information

For additional information on the unique abilities and features of Cadence Tensilica processors, refer to http://ip.cadence.com/ipportfolio/tensilica-ip.

References

1. Fusion G3 User's Guide
2. ConnX BBE32EP User's Guide
3. Johnson et. al., A Methodology for Designing, Modifying, and Implementing Fourier Transform Algorithms on Various Architectures, Center for Large Scale Computation, NY 10036, Sept. 1989.
4. Implementing FMCW Radar Signal Processing Algorithms on the Tensilica Fusion G3 DSP Application Note
5. Implementing FMCW Radar Signal Processing Algorithms on the Tensilica ConnX BBE32EP DSP Application Note