Implementing a Digital Monopulse Comparator in SIMULINK

Implementing a Digital Monopulse Comparator in SIMULINK

This report details the process of designing and implementing a digital monopulse comparator system using MATLAB's SIMULINK platform. The monopulse technique is widely used in radar applications to accurately estimate the direction of arrival (DOA) of signals and determine target positions through simultaneous processing of multiple signal channels.

Understanding Monopulse Comparator Systems

A monopulse comparator is a critical component in radar tracking systems that processes signals from antenna arrays to precisely locate targets. Traditional monopulse systems operate in the analog domain, but digital implementations offer advantages in flexibility, precision, and integration with modern signal processing systems.

Basic Principles

The monopulse technique uses four beams to measure the angular position of a target. All four beams are generated simultaneously, allowing differences in azimuth and elevation to be measured in a single pulse, hence the name "monopulse"[1]. The technique works by:

1. Receiving echoes from different elements of an antenna array
2. Processing these signals to generate sum and difference patterns
3. Using these patterns to estimate the direction of arrival (DOA) of signals[1]

A monopulse comparator network (also called an "arithmetic network") processes the four quadrants of a monopulse antenna into the SUM, Delta AZ (azimuth difference), Delta EL (elevation difference), and Delta Q signals[2]. While this processing can be done in the digital domain, traditional implementations often use analog components.

Required Tools and Toolboxes

To implement a digital monopulse comparator in SIMULINK, you'll need:

1. MATLAB and SIMULINK (base environments)
2. DSP HDL Toolbox (for implementing HDL-compatible digital signal processing components)
3. Fixed-Point Designer (for developing fixed-point algorithms optimized for hardware)
4. Phased Array System Toolbox (for modeling and simulation of phased array systems)[1]

5. RF Blockset (optional, for more advanced RF simulation including nonlinear effects)[3]

Implementation Steps in SIMULINK

1. Create a New SIMULINK Model

Start by creating a new SIMULINK model that will contain your digital monopulse comparator implementation. The model should be structured with clear input and output interfaces.

2. Design the Digital Down-Conversion (DDC) Subsystem

The DDC is essential for processing the received signals:

1. Create a subsystem that includes a Numerically Controlled Oscillator (NCO) block
2. Add low-pass filter blocks that support HDL code generation
3. Configure the NCO to generate signals that will mix and demodulate the incoming signals[1]

The DDC subsystem will convert the high-frequency received signals to baseband for easier processing. A typical implementation might sample at 80 MHz with a carrier frequency of approximately 15 MHz[1].

3. Implement the Digital Comparator

The digital comparator is the core of the monopulse system:

1. Create a subsystem for the digital comparator
2. Add blocks to calculate steering vectors for different elements of the antenna array
3. Implement sum and difference channel processing
4. Configure the subsystem to output sum, azimuth difference, and elevation difference signals[1]

The implementation should look similar to the one shown in the FPGA-Based Monopulse Technique example, where steering vectors are generated for specific incident angles (e.g., azimuth 30 degrees and elevation 20 degrees)[1].

4. Design the Monopulse Sum and Difference Channel Processor

This subsystem processes the downconverted signals with the steering vectors:

1. Create a subsystem for sum and difference channel processing
2. Add blocks to multiply the downconverted signals by conjugates of the steering vectors
3. Implement sum and difference calculations for azimuth and elevation
4. Configure the outputs to provide sum channel, azimuth difference, and elevation difference signals[1]

5. Use Fixed-Point Data Types

For hardware implementation, use fixed-point arithmetic:

1. Configure all blocks to use fixed-point data types
2. Use the Fixed-Point Designer toolbox to optimize the bit widths and scaling
3. Perform bit-true simulations to observe the impact of limited range and precision[1]

6. Verify Your Implementation

Compare your implementation with a behavioral reference model:

1. Create a floating-point behavioral reference model using the `phased.MonopulseFeed` System object
2. Compare the outputs of your HDL-ready implementation with the behavioral model
3. Calculate and analyze the error between the two implementations[1]

The model should have two branches: a behavioral floating-point model at the top and a fixed-point HDL-compatible implementation at the bottom, allowing for direct comparison of outputs[1].

Example Implementation in SIMULINK

The following steps provide a more detailed workflow based on the FPGA-Based Monopulse Technique example from MathWorks:

Setting Up the Input Signal

Assume a 4-element Uniform Rectangular Array (URA) receiving signals:

1. Create four input channels representing signals from each element of the URA
2. Configure the input signals with appropriate sampling rate (e.g., 80 MHz) and carrier frequency (e.g., 15 MHz)[1]

Implementing the DDC Module

The DDC module consists of:

```
open_system([modelname '/DDC and Monopulse HDL']);
```

1. NCO block for signal mixing
2. Low-pass filters to complete the downconversion
3. Appropriate delays to compensate for the downconversion chain[1]

Creating the Digital Comparator

The digital comparator compares the steering vectors:

```
open_system([modelname '/DDC and Monopulse HDL/DIGITAL COMPARATOR']);
```

1. Generate steering vectors using the `phased.SteeringVector` System object
2. Implement the comparison logic that produces the desired sum and difference outputs[1]

Implementing Sum and Difference Channel Processing

The sum and difference channel processor:

```
open_system([modelname '/DDC and Monopulse HDL/Monopulse Sum and Difference Channel']);
```

1. Takes the compared steering vectors
2. Calculates the sum channel, azimuth difference, and elevation difference
3. Uses fixed-point arithmetic throughout the processing chain[1]

Verification and Performance Analysis

To verify your implementation:

1. Compare the sum and difference patterns with theoretical expectations
2. Analyze the null depth, which defines angular resolution and target tracking accuracy
3. Evaluate the performance by varying the angle of arrival and measuring the results on the output ports[3]

For optimal performance, ensure that:

- Phase differences between antenna ports are properly compensated
- The null depth is sufficiently deep (ideally 40+ dB lower than the sum signal)
- The system provides accurate angle measurements across the required field of view[3]

Conclusion

Implementing a digital monopulse comparator in SIMULINK involves creating a system that processes signals from multiple antenna elements to accurately determine a target's direction. By using SIMULINK's block-based modeling approach and MATLAB's specialized toolboxes, you can design, implement, and verify a digital monopulse comparator that can be later translated to hardware using HDL code generation.

The advantages of a digital implementation include flexibility in algorithm design, ability to perform bit-true simulations before hardware implementation, and easier integration with modern digital signal processing systems. However, careful attention must be paid to fixed-point arithmetic, timing, and phase relationships to ensure accurate angle estimation.

For further refinement, consider importing measured S-parameters from actual antenna and comparator hardware to improve the fidelity of your simulation[3], or exploring different comparator designs using various coupler configurations[2].

⁂

1. https://www.mathworks.com/help/dsphdl/ug/fpga-based-monopulse-technique-workflow-part-1.html
2. https://www.microwaves101.com/encyclopedias/monopulse-comparator-networks
3. https://www.mathworks.com/help/simrf/ug/design-and-simulation-of-monopulse-tracking-system.html