An FPGA based processor yields a real time high fidelity radar environment simulator

R.J. Andraka[1], and R.M. Phelps[2]
[1]Andraka Consulting Group, Inc. 16 Arcadia Drive, North Kingstown, RI 02852
[2]Telephonics Corp., 815 Broad Hollow Road, Farmingdale, NY 11735

Abstract

Radar parametric testing has traditionally required either expensive trials in real-world situations (with many uncontrolled variables) or very limited 'canned' tests. The Von Neumann processors normally used for signal processing are severely limited in this application because of the inherently serial instruction stream. This paper discusses the use of FPGAs to accelerate the processing to obtain a near real-time environment simulator. The FPGA logic handles the time sensitive tasks such as target sorting, waveform generation, sea clutter modeling and noise generation. DSP microprocessors handle the less critical tasks like target movement and radar platform motion. The result is a simulator that simultaneously produces several hundred independent moving targets, realistic sea clutter, land masses, weather, jammers and receiver noise.

I. INTRODUCTION

A. Traditional Test Scenario

Traditional Radar Target Generators (RTGs) are built to test, debug, and demonstrate radar and target tracking functions. Their aim is not a realistic simulation of the radar environment, but rather to perform some basic testing. The number of targets is limited and target motion is simple. The radar platform doesn't change position or attitude. Interference is simulated by a simple Gaussian noise generator. The test scenarios are canned; there is no real time interaction between the radar/operator and the RTG

With traditional RTGs, it is not possible to adequately evaluate system performance measures such as probability of false alarm (P_{fa}), probability of detection (P_{det}), and tracking accuracy (σ_x, σ_v) in realistic environments. The simple traditional test scenarios do not reflect real radar operation. Realistic radar missions are not simulated or evaluated

B. LAMPS MMR Radar Target Generator

The LAMPS Multi-Mode Radar (MMR) Secondary Equipment Radar Target Generator (RTG) developed by Telephonics breaks the mold of the traditional RTG. The number of targets is in the hundreds. Target and platform (helicopter) motion are complex and realistic. High fidelity clutter and noise generators are used. With realistic interference simulation, it is possible to evaluate P_{fa}, P_{det}, and Track accuracy. The LAMPS RTG can be run with realistic missions and real time interaction.

The high performance LAMPS RTG was designed to perform much of the qualification and acceptance testing of the LAMPS MMR, and to be integrated into a larger interactive LAMPS scenario generator with a human operator, helicopter pilot, and other LAMPS sensors.

1) Radar Performance Testing

One of the purposes of the LAMPS RTG is to perform much of the qualification and acceptance testing of the LAMPS Multi-Mode Radar (MMR). This testing would be minimally supplemented by real-world testing (shore tests, flight tests). Testing evaluates both user performance requirements and derived system requirements.

User Performance Requirements include P_{det} and track accuracy for specified targets (radar cross section, distance, height), interference (sea clutter, rain, sidelobe jamming), detection method (manual display, Track While Scan) and radar mode. For track accuracy, targets are linear or maneuvering, and the helicopter moves within specified limits for attitude (roll, pitch, and yaw), and attitude change rates. P_{fa} levels are specified for interference, detection method and radar mode. System Requirements are derived from the User Requirements. They specify design decisions, algorithms, and implementation details. All of these are tested using the LAMPS RTG

Some performance requirements such as P_{fa} are impossible to test in the real world: how would you know that detected targets are not real, e.g., flying birds or floating beer cans. Track accuracy can be tested in the real world only to the accuracy of the instrumentation aboard the target. Detection is probabilistic, requiring numerous runs to test each specified test condition.

A feasible testing methodology is to run all performance tests with the LAMPS RTG, and selected real world tests. Agreement (consistency) between the real world tests and the RTG tests validates the RTG simulations.

2) LAMPS RTG Requirements

The performance requirements for the LAMPS RTG come from its required use in testing the LAMPS MMR and reacting in real-time to operator commands.

To test the LAMPS radar, the RTG must be capable of generating data in all radar modes. To evaluate tracking accuracy, the RTG target position accuracy is required to be an order of magnitude better than MMR accuracy specifications. To test target detection, the target and interference signal levels must be accurate to a fraction of a decibel. To test false alarm rates requires noise and clutter generators to have accurate tail distributions at P_{fa} levels of 10^{-5}.

The real time operation requirement precludes the possibility of elaborate canned scenarios, even with clever preprocessing. Calculations must be done on the fly – data must load quickly.

II. Model Development

A. Target Models

Two variations of the target model are used, depending upon the tests performed. An uncompressed model generates realistic uncompressed video for point targets. It is used to test the MMR signal processing, and to evaluate the radar's ability to resolve targets. A compressed model simulates the video signal as it appears after being compressed by a matched filter. Since the sea clutter model is valid only for the compressed model, the compressed model is used to evaluate the probabilities of false alarm (P_{fa}) and target detection (P_{det}) and to verify tracking accuracy.

Regardless of the model, the simulator must be capable of producing several hundred targets, each with an independent motion trajectory, in order to be useful in the LAMPS mission simulations. Motion of the radar platform (a helicopter) in 6 axes modulates the apparent target motion and position. There may be as many as 100 targets at any given azimuth, so the processor must be able to handle up to 100 targets in a single PRI (pulse repetition interval). Targets whose responses overlap in range create a special challenge for the generator models chosen. A compromise was reached permitting no more than two target responses to occupy any range cell to avoid unnecessarily complicating the hardware. A two target overlap is sufficient to test the resolution of closely spaced targets. This limitation has minimal impact on the system simulation.

1) Uncompressed Target Model

Target video received at the radar is the reflected energy of the radar pulse transmitted, delayed by the time the signal takes to propagate to the target and back. Without pulse compression, the range resolution of radar is limited by the width of the transmitted pulse. The minimum detectable target and the maximum range are a function of the power in the transmitted pulse. In order to keep the peak power at practical levels and still be able to detect targets at reasonable ranges, the pulse width must be widened. This deteriorates the range resolution. By transmitting a signal that changes as a function of time, then passing the received energy through a matched filter, a finer range resolution is achieved while spreading the transmitted pulse power out over time. This is known as pulse compression in radar. The LAMPS radar's transmitted pulse frequency is linearly swept, resulting in a linear chirp. The received energy for each point target is also a chirp with the same characteristics as the transmitted chirp.

When the chirp is sampled, as it is in the radar receiver, the start frequency of the chirp appears to be linearly modulated by the timing relationship between the start of the chirp and the instant the sample is taken. The time relationship is a function of the range of the target. Therefore, by modulating the start frequency of the sampled chirp, the simulator can accurately position the target at fractional range bins (i.e. between samples). A sampled target chirp can be produced by a digital complex sinusoidal oscillator with the appropriate start frequency and frequency sweep rate. A numerically controlled oscillator (NCO) accumulates phase increments to generate a rotating phasor. By modulating that phase increment, a linear chirp with the appropriate start frequency and chirp slope is created. The NCO is reset between targets so that the chirp modulation is properly initialized. The starting phase of the NCO for each target should be random so that closely spaced targets are not coherent.

Overlapped targets result in superposition of the chirps from each target. To accurately simulate this, an additional NCO is required for each target simultaneously producing a chirp. In order to handle two overlapped targets, the simulator uses a pair of complex NCOs, each with it's own start frequency register. A target with a large range extent can be represented by a large number of point targets closely spaced. Since the chirps from each point will overlap, it is clear that the NCO is only sufficient to model point targets.

2) Compressed Target Model

The compressed target model represents the target video after it has been compressed by a matched filter. Matched filtering creates a pulse corresponding to the target position. The uncompressed video is the convolution of the target with the chirp waveform. The matched filter essentially deconvolves the chirp to recover the target pulse. Since we are generating the target, the same effect is attainable by filtering the target with an FIR filter whose coefficients are chosen to provide the matched filter response for each fractional range cell offset. Since the coefficients depend on the target's position relative to the range cell boundaries, separate generators are required for each target simultaneously (in the same range cells) generated. Again, a compromise was reached permitting no more than two overlapping targets to keep the hardware reasonable.

B. Interference Model

Radar interference is the sum of several noise and clutter sources. Sea clutter is generated using the K-distributed sea clutter model [1]. The sea clutter is combined with receiver noise, weather, landmass targets and jammers to obtain a composite interference as shown in Figure 1 and described in Table 1. The noise and clutter from each source are uncorrelated, so they are combined using root sum of squares. The combined noise and clutter magnitude profile is used to modulate a complex Gaussian noise source to generate the total interference. The interference has to be modeled so that the tail probabilities are accurate to 10^{-6} to facilitate testing of P_{fa} and P_{det}.

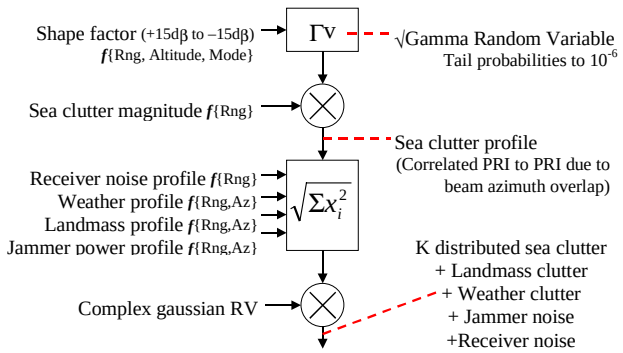

Figure 1. Interference model

Sea clutter is modeled using the K-distribution described by Ward et al [1] with parameterized shape factor and clutter power. Clutter power is modeled using a constant gamma reflectivity model. The K-distribution clutter model offers several advantages over other sea clutter models: a) It models sea clutter statistics accurately for a variety of radars and sea clutter conditions. b) The shape factor parameter creates a diverse family of clutter distributions: Small values generate large sea spikes (the distributions have large tails), large values generate benign sea conditions (small tails). c) The shape factor parameter is defined in terms of the radar's parameters (range/angle resolutions, and polarization). Fitting the model to a particular radar is straight forward - it doesn't involve making empirical measurements with the radar. d) Using the K distributed model in the RTG provides the flexibility to modify the model (by adjusting the shape factors and reflectivity) without changing the hardware.

The K-distribution is the product of a gamma random variable and a Rayleigh random variable. (The K-distribution complex voltage is the product of a square root gamma and a complex Gaussian random variable.) The gamma random variable varies from scan-to-scan; with frequency diversity, the Rayleigh random variable varies pulse-to-pulse. The K-distribution is defined by two parameters: shape factor and power. Shape factor is a function of grazing angle, range and cross-range resolution, wind/ swell direction, and polarization. Power depends on the clutter reflectivity. Using the constant-gamma model, clutter reflectivity is a function of sea state and grazing angle.

Radar interference is modeled as the sum of Gaussian Noise and K-distributed Clutter. This interference model is defined in Table 1. A formula for the shape parameter (empirically derived in Reference [1].) is given in Table 2. The constant-gamma clutter reflectivity model is defined in Table 3.

Clutter and noise interference is generated using a square root of Gamma Random Noise Generator (RNG) and a Gaussian RNG as described in Table 4.

Table 1 Interference model: K-distributed clutter + noise

I = Interference = $N \cdot \sqrt{P_n + P_c \cdot G}$ where P_n = Noise Power, P_c = Sea Clutter Power
G = Gamma Distribution (for mean = 1) density: $f(x) = [\alpha^{\alpha} / \Gamma(\alpha)] * x^{\alpha-1} * e^{-\alpha * x}$, $x > 0$ parameter: α = shape factor
N = Complex Gaussian Noise = X + iY where X, Y are normal distributed, mean = 0, $\sigma^2 = .5$

Table 2 Shape parameter for K-distributed clutter

α = **Shape Parameter** **= 10^ (2/3 log Ψ + 5/8 log A + σ + k - .9042)**
Ψ = grazing angle (radians) A = area (meters^2) = range resolution x cross-range resolution σ = { - 1/3 for up or down swell direction; 1/3 for across swell directions; 0 for vertical polarization or when no swell exists **k** = { .7 for vertical polarization; 0 for horizontal polarization

Table 3 Clutter reflectivity: constant gamma model

Reflectivity = γ * sinΨ γdB = -25.05 + 6 * (Sea State - 3) Ψ = grazing angle = asin(h/r*(1+h/(2*re))-r/(2*re)) *(exact curved Earth formula)*
Clutter Cross Section = Clutter_Patch * Reflectivity = (r*Δθ*Δr/cosΨ) * (γ*sinΨ)

Table 4 Implementation of interference model

Interference = $\sqrt{P_n^2 + P_c^2 \cdot g^2} \cdot (X + Y \cdot i)$ where g = √G, from square root of gamma RNG X, Y from Gaussian RNG

The square root gamma random variable is a seven bit fixed-point number. The RNG uses the inverse histogram method to obtain a 7 bit square root gamma distributed random variable (128 probabilities to define the cumulative distribution: a 32-bit uniform random number is generated,

and a 7 iteration binary search is performed). The gamma random variables are generated for a wide range of discrete shape factors: -15, -14.75. … , 14.75, 15 dB. Each shape factor requires it own cumulative distribution probabilities. The inverse histogram method is shown graphically in Figure 2.

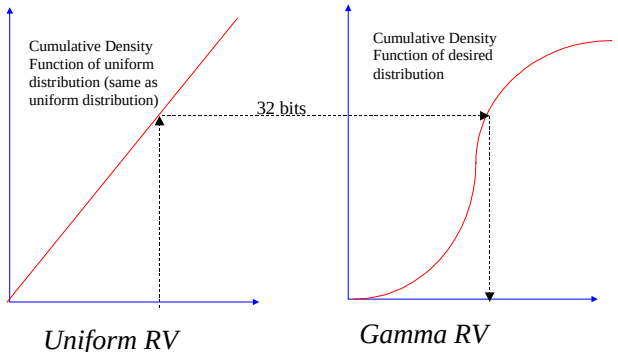

Figure 2 Inverse histogram method used to generate gamma random variables from uniform random variables

The gamma random variable distribution functions for the shape factors 15, 10, 5, 0, -5, -10, and –15 dB are plotted in Figure 3. Sea spikes (very large clutter returns) are generated when the tails of the gamma distribution are large (i.e., for the smaller shape factors).

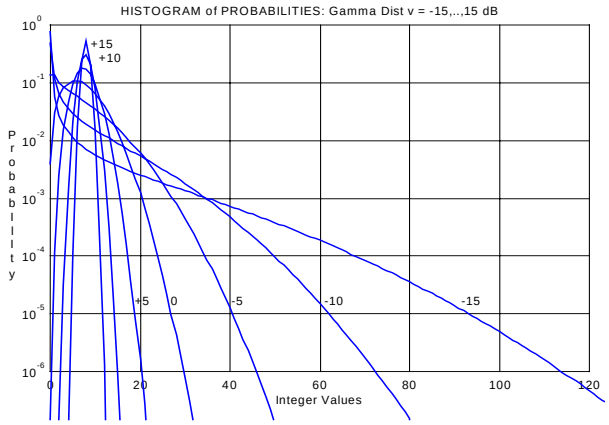

Figure 3 RTG Square Root Gamma Densities

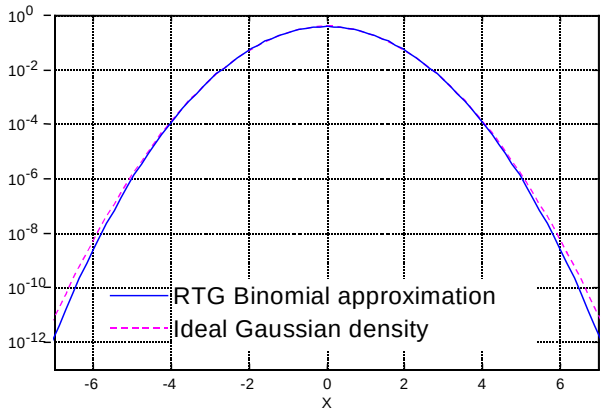

Figure 4 RTG Gaussian Density

Gaussian random numbers are generated by a binomial approximation using 128 binary events (the sum of 128 1 bit uniform random variables with the result normalized). Tail distributions obtained by this generator are faithful to the gaussian ideal to probabilities of 10^{-6} (see Figure 4)

The square root gamma random variables in the sea clutter model are correlated PRI to PRI because of beam overlap at adjacent azimuths. To compute the degree of correlation, a rectangular beam model was used, and based on the PRI and azimuth rate, the number of PRI's contained within the beam width was determined. This directly translates to a percentage change from PRI to PRI. For the MMR, the maximum replacement rate is 25% based on the beam width, PRI and azimuth rate.

Azimuth clutter power correlation for different numbers of hits (PRIs) in a rectangular beam are plotted in Figure 5. The clutter is K-distributed with a small shape factor and frequency diversity. The initial drop of 0.5 results from frequency diversity (the Rayleigh part of the random variable decorrelates PRI to PRI.) Linear correlation and finite extent result from the rectangular beam shape.

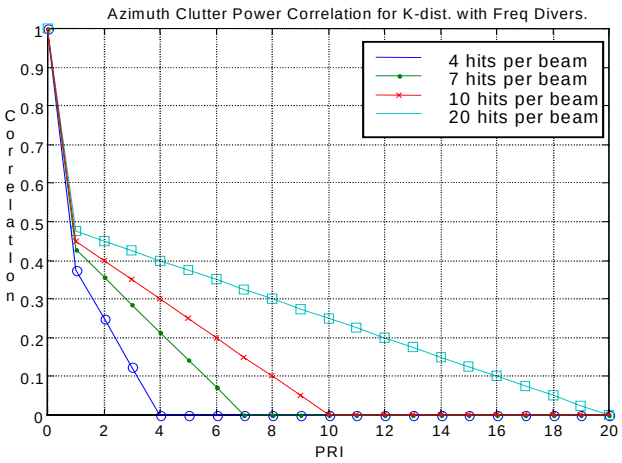

Figure 5. Sea clutter power correlation due to beam overlap

Using the current gamma values to obtain the next value through some formula would obtain the needed correlation, but has the undesirable side effect of tainting a non-Gaussian probability distribution. Instead, we can retain the clutter values for each range cell from PRI to PRI, randomly replacing values so that the proportion of changed cells reflects the expected change due to beam movement. This provides the desired correlation from PRI to PRI without upsetting the probability distribution of the sea clutter. Randomly selecting range cells results in some cells being replaced more than once per PRI, while other cells aren't replaced for many PRI's. This differs from reality; the beam sweep causes all cells to be replaced after the number of PRI's contained in the beam width. The solution is to select the range cells in a random sequence without revisiting any cell until the sequence is complete. The sequence is permuted each time it starts over to maintain the apparent randomness. In addition to being more a faithful reproduction, this method has the advantage of requiring a lower replacement rate than random replacement. Random replacement requires a higher rate to reduce the correlation beyond the beam width and to compensate for cells that are changed more than once per PRI.

III. IMPLEMENTATION

A. Task Allocation

The Radar Environment Simulator is controlled by, and is partially implemented in software. Tasks that are not time critical are handled by software. Time critical tasks are performed by custom hardware.

The radar sweeps in azimuth. At each incremental azimuth, it sends a linear chirp and receives the reflected energy. The time interval between the beginning of successive chirps is the pulse repetition interval (PRI). Each PRI is divided into an integer number of range bins representing the reflected energy at successive range intervals. Each range bin contains a digitized complex sample of the reflected waveform.

For each PRI, simulated targets at the current azimuth need to be sorted in range order. Targets outside of the active ranges (determined by the timing of when the receiver is enabled relative to the transmitted pulse) are discarded. Waveforms need to be generated for targets within the active ranges. For the uncompressed model, the target waveforms are linear chirps, while for the compressed model the targets are created by running a rectangular target pulse through an FIR filter to simulate a chirp compressed by a matched filter. This sorting and waveform generation is accomplished in hardware, as it is too complex to be handled by software within the time of a PRI.

Targets are slow moving relative to the PRI and azimuth sweeps, so the target motion is handled by software, as is the sorting of targets by azimuth. For each PRI, the software presents the hardware with an unsorted list of targets to be generated for that PRI. Each target is represented by a 64 bit target descriptor containing a 16 bit starting range (range where the target first appears), a 16 bit range extent (the number range cells occupied by the target response), the magnitude of the target, and a waveform characteristic. For the uncompressed model, the waveform characteristic is the chirp start frequency. Carefully chosen variations in the start frequency have the effect of moving the target by a fractional range bin. In the compressed model, the waveform characteristic is a code to select one of ten filter coefficient sets. Each set is computed for one of ten fractional range offsets. This again permits the target to be positioned at fractional range offsets. A special PRI terminator marks the end of each PRI's unsorted target list so the hardware knows which targets belong to which PRI.

The sea clutter is random noise with a specific distribution and with correlation from PRI to PRI. To obtain the correlation and the required rate of change, the sea clutter is generated in hardware using range attributes provided by the software (shape factor and magnitude by range and change rate). The sea clutter generator updates a sea clutter profile table in a random sequence. The table is read linearly once for each PRI to obtain the sea clutter magnitude at each range.

The Gaussian interference, consisting of the sum of weather, land mass, and jammer returns is relatively static. These values are maintained in an azimuth map in software. For each PRI, the software presents the hardware with a range dependent noise profile representing the root sum of squares combination of these noise magnitudes. The hardware combines the values of the noise for each range with the sea clutter profile and the receiver noise profile using root-sum-of-squares. The combined profile is the total interference noise magnitude at each range. This profile modulates a complex Gaussian noise source in the hardware to generate the interference.

Software presents the Gaussian noise profile and the sea clutter range tables in compressed format in order to conserve the bus bandwidth. Decompression of these tables is done by the hardware. Gaussian noise profiles are represented as quadratic polynomial segments. The software provides the start range and coefficients for each segment. Polynomial expansion is performed by the hardware to recover the profile. The sea clutter parameter tables are compressed using run-length encoding which is decoded in hardware to recover the table contents.

The required hardware consists of a ping pong input target buffer, target sort & waveform generation, sea clutter generation and table, Gaussian clutter expansion and the complex Gaussian noise source. The hardware also includes logic to combine the target and interference waveforms, and to limit, format and buffer the output.

B. A Custom Hardware Solution

The original concept separated the hardware functions into three dedicated-function custom modules: A target generator, a clutter generator, and the PRI buffer. Two sets of these modules were to be used in the complete RES system to realize the required throughput.

The target generator module was to perform the input buffering, target sorting and target waveform generation for both the compressed and uncompressed models. Uncompressed waveform generation was to be done using a pair of STEL 1180 NCO's [2] and a pair of 12x12 bit multipliers for each of two target channels[1]. Compressed targets were to be produced using a pair of Harris FIR filter chips. The approach to sorting the targets was unresolved, with a scheme using Content Addressable Memories (CAMs) from Music Semiconductor high on the list of proposed solutions. Each of these special purpose chips requires a special interface to set the function and parameters. Preliminary estimates indicated that the target module logic might not fit on a single board.

The proposed clutter module was to generate new sea clutter values for each range cell on each PRI, then randomly

[1] a pair of real-only NCOs is required to generate the quadrature chirp, and the multipliers are needed to scale the complex chirp by the target magnitude

use either the new value or the one from the previous PRI. That module also was to decompress the Gaussian clutter, generate the complex Gaussian noise and combine the resulting interference with the target waveforms from the target module.

An additional module, the PRI buffer was proposed to limit the combined waveforms, format them into either IQ data or magnitude data (depending on radar mode) and buffer the data by PRI in a ping pong buffer.

C. An FPGA Solution

The original concept design presented too many design challenges without the added pressure of an aggressive schedule and an incomplete design specification. Custom fixed function hardware effectively locked the algorithm in the board design, leaving no options if flaws were later found in the algorithms or specifications (and there were some). The RES development paralleled the Multi-Mode Radar (MMR) development. Changes in the radar rippled over to specification changes in the RES. The changes were too frequent to be able to proceed comfortably with an unchangeable RES design. Additionally, we identified a potential real-estate shortage on the target generator module, and later a slot shortage in the system rack. As a result of these potential problems, we pushed for and obtained reluctant approval for an FPGA based system.

Using FPGAs allowed us to eliminate all of the special purpose chips and obtain a greater level of integration in the design. We immediately recognized that a common board design could be used for all the modules in the system if the required logic and interconnect for all applications existed on the board. We studied the hardware requirements to arrive at a common board architecture. First a block diagram, showing all the components of the target, clutter and PRI buffer modules, was drawn. We tailored each sub-function for implementation in an FPGA or as a memory lookup, then repartitioned the design into two boards. Each board was trial-partitioned into FPGAs. The resulting partitioned block diagrams were overlaid on one another and then adjusted to reuse memories and interconnect as much as possible. After several iterations, a minimum common architecture became apparent. We then expanded that architecture to complete the symmetry and make all connections the same for each FPGA device. The resulting architecture is a linear systolic array of four 4025E-2 FPGAs. The systolic connections are 32 bits wide. Each device has a pair of completely independent 64K x 16 12ns SRAMs attached to it on private busses. Each FPGA also connects to either of two auxiliary busses by way of bus switches. The auxiliary busses provide general interconnect between the FPGAs. Additional bus switches connect two expansion ports to the dangling systolic busses or to one of the auxiliary busses. The architecture of the module, dubbed the Flexible Pipeline Processor, is shown in Figure 6.

The module includes an FPGA configuration controller and EEPROM storage for 16 FPGA programs. The host software can direct configuration of all 4 FPGAs with a single word write (four 4 bit fields, one for each FPGA select which of the 16 stored programs is loaded into each FPGA). The host interface and configuration controller are implemented in Lattice Semiconductor In-System Programmable (ISP) logic. Non-volatile programmable logic here provides a permanent interface with the flexibility to make changes if necessary. (The Pentek MIX interface varies somewhat from the Intel MIX interface standard, and at the time all of the details of those variations were not available, so programmable logic provided a hedge against uncertain bus specifications).

The repartitioning of the design, along with the higher integration achieved by using FPGAs to perform the special functions allowed us to include the PRI buffer function on the target generator module.

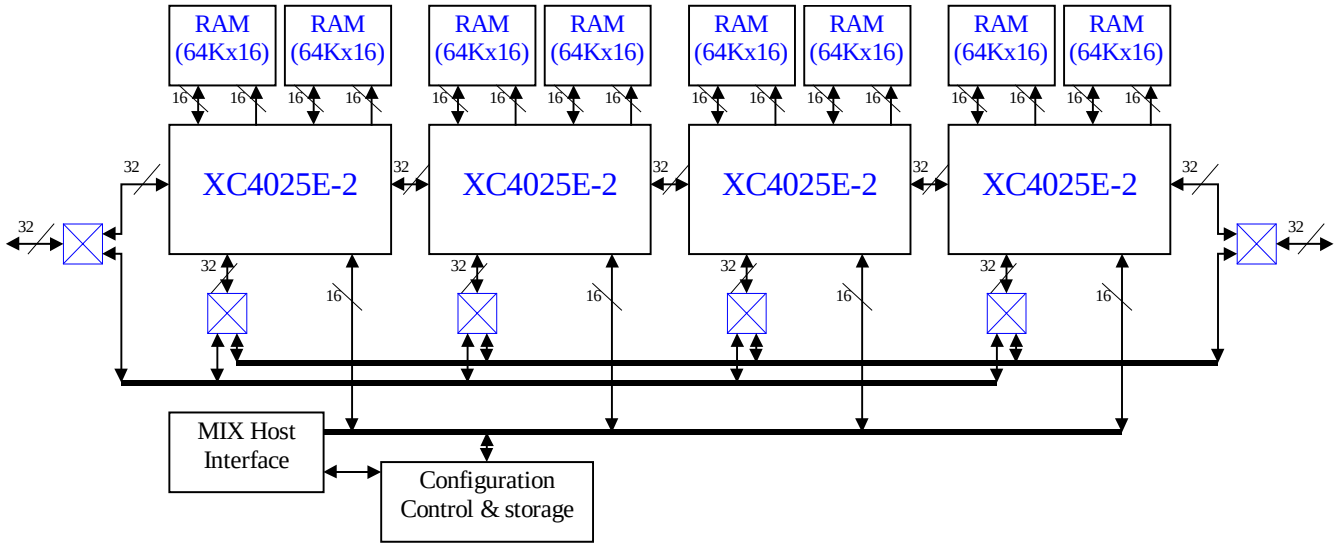

Figure 6. Flexible Pipeline Processor board architecture

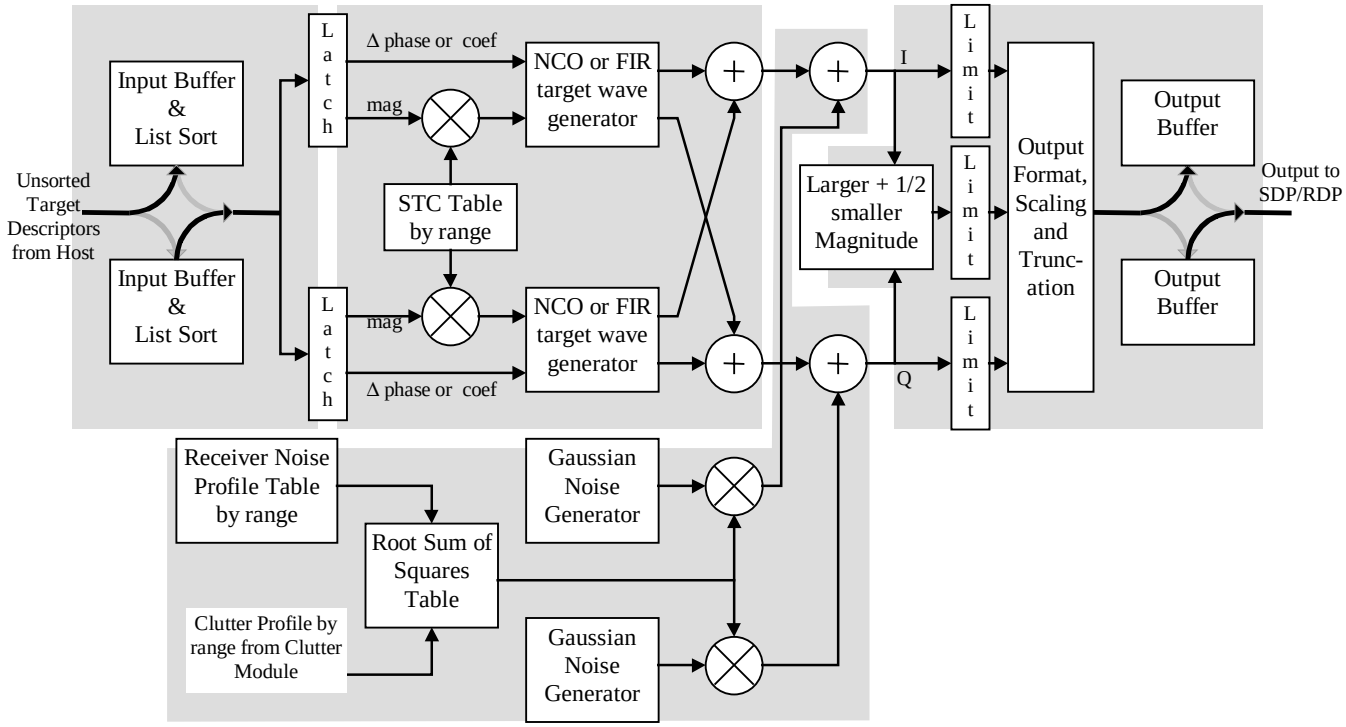

Figure 7. Target generator block diagram. The uncompressed model uses NCOs while the compressed model uses FIR filters with Q =0. Shading shows the FPGA partitioning

D. Target Generator Design

The Target Generator is a set of FPGA programs for the Flexible pipeline processor module. The target generator takes advantage of the FPP reconfigurability to handle the different target generator models, as well as to provide the host access for module set up and function test. Figure 7 shows the target generator architecture for both the uncompressed and compressed target models. The uncompressed model uses the NCO generator, while the compressed model uses the FIR generator. The target generator accepts an unsorted list of targets from the host for each PRI. It sorts the list then generates target waveform at each range cell. Two target waveform generators are used to permit overlapping targets. Interference corresponding to the combined clutter and receiver noise is summed with the target responses. The resulting data is limited, scaled, formatted and stored in a ping pong buffer.

1) Target Sorting

Target data for each PRI is written to the module by the host software as an unsorted list of target descriptors terminated with an end of list write. Each descriptor consists of a start range (first range bin the target appears), a length (number of range bins the target appears in), the target magnitude, and a target characteristic. The characteristic, which sets the fractional cell range, is the starting frequency of the chirp for the uncompressed model or the filter coefficient select code for the compressed model.

The target sort is accomplished using a tag memory and a register file. The length, magnitude and characteristic for each descriptor are written to a register file in the order they are presented by software. The register file pointer is written to a 64K deep tag memory at the address equal to the start range parameter in the target descriptor. The memory is presumed to be clear before writing any tags. When the target list is complete (end of list marker written) and the output buffer is available for a PRI of data, the tag memory is read in range order. Each time a non-zero tag is encountered, the contents of the register file corresponding to that tag are fetched and presented to the waveform generator. Each location in the tag memory is cleared immediately after it is read so that upon reaching the end of the PRI, the tag memory is clear. Two such buffers are used so that one is accepting the next PRI's list while the other is outputting a PRI's sorted data. The sorted targets are latched as they are read out so that they persist for the number of range bins specified in the target's descriptor. The target sort algorithm allows targets in closer ranges than the first range of the PRI to be evaluated and processed so that in-range 'target tails' appear even though the start range of those targets is outside the active ranges. The target sort logic accepts up to 127 targets per PRI, including those not in active ranges (active ranges are those between the PRI start range and PRI range extent).

2) Target Waveform Generation

The sorted target descriptors feed one of a pair of target waveform generators. In the compressed model, each target generator is essentially an FIR filter, while in the uncompressed model each generator is a numerically controlled oscillator (NCO) with a complex output proportional to the target magnitude. The FPGA is reprogrammed with the appropriate generator when the models are changed.

In both the compressed and uncompressed cases, there are two instances of the appropriate generator so that overlapped target responses may be produced. The IQ responses from the two generators are combined using the vector sum.

The NCO used for the uncompressed model consists of a phase accumulator capable of producing linear chirps and a polar to cartesian converter. The polar to cartesian conversion translates the phase angle and the target's magnitude to the I and Q components of the complex chirp. The phase accumulator integrates a constant chirp slope (programmed by the host), adds the resulting phase increment to the start frequency defined by the target descriptor and integrates the sum to create the phase angles for successive samples of the chirp. The phase angle is randomized when the accumulator is not producing a target.

A CORDIC rotator converts the phase and magnitude representation of the chirp to an IQ format by rotating a vector representing the target magnitude from the I axis through the angle provided by the phase accumulator. CORDIC rotation is a shift-add algorithm for rotating vectors [3]. The CORDIC implementation has about the same logic complexity as a single multiplier, yet replaces two multipliers and sin/cos look up tables.

The target magnitudes are scaled by a static range-dependent STC curve to simulate range attenuation. The magnitudes are multiplied by values from an STC curve table before being passed to the NCO (STC curves are not applied by hardware for the compressed model). The STC curve table is stored in one of the memories associated with the waveform generation FPGA. Since the table is static, it is loaded before the NCO design is loaded, eliminating the load logic from the design. The STC curve also has a bit that causes targets to be blanked (suppressed) in ranges where it is set to simulate receiver blanking.

The compressed model uses an FIR filter for each waveform generator. The filter coefficients are selected to generate the desired (real only) target waveform from a rectangular input. The filter input is the magnitude from the target descriptor, which persists for the number of range bins specified by the descriptor's length parameter. The filter is substantially simplified by generating the response to a normalized target and multiplying the resulting waveform by the target magnitude (this eliminates the multipliers in the filter). The filter coefficients are selected from one of ten coefficient sets corresponding to equally spaced target position offsets into the range cell. The filter coefficients are written into registers in the FPGA by the host software, rather than being hard-coded in the design to avoid having to recompile the design for coefficient changes.

3) Gaussian Noise Generation

Interference is generated using a complex Gaussian noise generator modulated by the combined noise and clutter magnitude profile. The modulation is accomplished by multiplying the combined profile by each component of the complex Gaussian noise. Each Gaussian noise component (I and Q) is generated by summing 128 random binary values and a bias constant (bias shifts the mean to zero). The random binary values are produced by 64 independent 129 bit Linear Feedback Shift Register (LFSR) counters, each of which is randomly seeded. The 129 bit LFSR counters produce a pseudo-random sequence that takes over 3000 years to repeat when clocked at 80 MHz. Randomly seeding the counters assures (with reasonable certainty) that the counter sequences are independent over intervals of a few PRIs. A tally adder sums one bit from each LFSR to produce a 6 bit binomial random variable on each cycle of the 80 MHz clock. Two consecutive binomial random values are summed to obtain the 7 bit Gaussian approximation at 40 Mhz. The circuit is duplicated (but using different seeds) to produce an independent Q component.

The I and Q Gaussian noise generator outputs are each multiplied by the combined clutter profile. That complex product is summed (vector sum) with the combined target waveforms before they are limited, scaled and formatted for output. The combined clutter profile is the root-sum-of-squares combination (to properly combine the uncorrelated noise magnitudes) of receiver noise, land mass targets, weather, jammers, and sea clutter. The receiver noise profile is a static range dependent table programmed during system initialization by the host software (it is not accessible when the operational design is loaded). The hardware handles the receiver noise to lighten the software load and so receiver blanking is simulated correctly. The receiver noise profile is combined with the composite clutter profile from the clutter module using a root-sum-of-squares look up table. That composite profile modulates the Gaussian noise.

4) PRI Buffer

The combined target and noise waveforms are subjected to a programmable saturation limiter. The limiter independently limits the I and Q components of the composite signal to simulate clipping in the receiver channels. Each limiter is symmetric about zero. The limit value, common to both the I and Q channels, is programmed by the host software. In RTG modes where the module outputs magnitude instead of IQ data, the vector magnitude is computed using a larger-plus-half-smaller magnitude approximation before limiting. Larger-plus-half-smaller magnitude is used to match the magnitude function in the MMR radar equipment.

The output of the limiter is scaled by a software controlled power of two using a barrel shifter to set the system gain. The

output from the scaling logic is truncated to 8 bits (16 bits for 16 bit magnitude modes). The composite waveform is effectively dithered by the Gaussian noise before truncation.

The scaled result is stored in a ping pong buffer in range order. The buffer 'buckets' change input and output roles after a complete PRI is processed by the target generator and the previous PRI has been output to the radar.

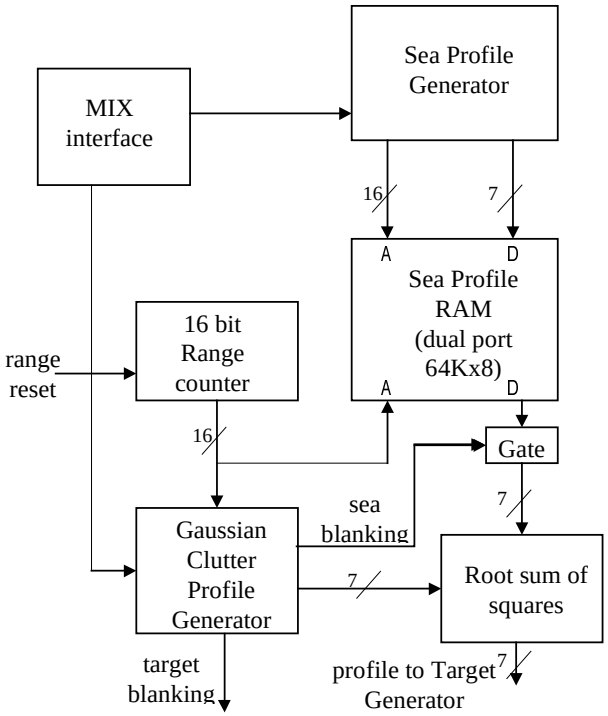

Figure 8. Clutter generator block diagram.

E. Clutter Generator Design

The Clutter Generator is a second set of programs for the Flexible Pipeline Processor. The clutter generator function is the same for all modes of the RTG. The sea clutter model is, however, accurate only for the compressed model. The clutter generator consists of a sea clutter profile generator and RAM, Gaussian clutter profile expansion logic, and a root-sum-of-squares function to combine the profiles. Figure 8 is a block diagram of the clutter generator module.

1) Gaussian Clutter Expansion

Gaussian clutter is the combination of clutter contributions from jammers, weather and land masses. Host software combines the noise magnitudes from these relatively static sources and stores the profiles in an azimuth map. The profiles are transferred to the clutter generator hardware for each PRI. Profiles are stored and transferred in compressed form so that software can transfer the data for all the range cells of a PRI within the PRI time (if not compressed, the software would have to transfer profiles at 40 Mbytes/sec just to keep up with the PRI processing). The compression scheme breaks the profile into a series of quadratic polynomial segments. Only the endpoints and the quadratic coefficients are transferred to the hardware. The hardware reconstructs the profile on the fly using the endpoints and coefficients supplied by the software. Each segment also carries two tag bits that specify whether sea clutter or targets are to be blanked during the ranges corresponding to that polynomial segment. This capability permits accurate modeling of landmasses and the ability to modulate the receiver blanking intervals on a per PRI basis (needed to support frequency diversity). The polynomial endpoints and coefficients are queued on the module in compressed form, where they are retrieved when needed by the expansion logic. The queuing permits polynomial segments to be stored several PRIs ahead of when they are actually needed.

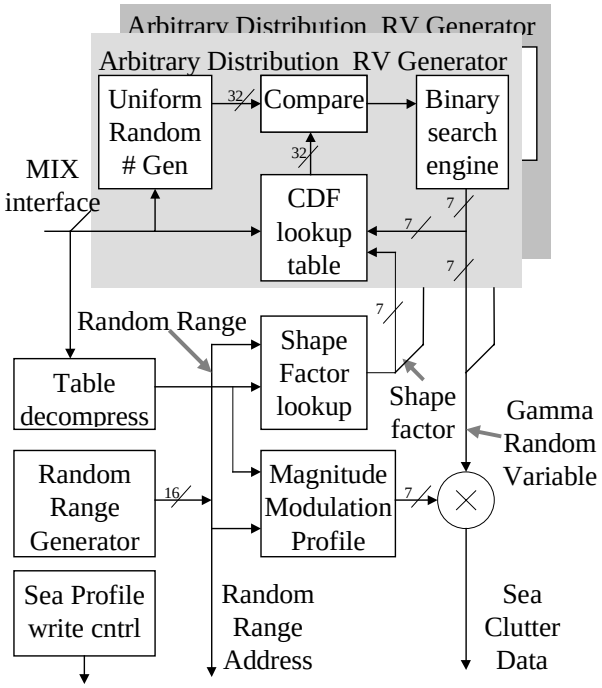

Figure 9. Sea Profile Generator Detail

2) Sea Clutter Generation

The sea clutter generator, shown in greater detail in Figure 9, updates a sea profile table in random range order. That table is read out in range order for each PRI to obtain the current sea clutter profile. The value at each range cell in the sea profile table is a gamma distributed random value scaled by a range dependent magnitude value. Software computes the magnitude based on the constant gamma clutter reflectivity model and updates the table (organized by range) in hardware whenever there are platform altitude, sea state or radar mode changes. The shape factors for the gamma distributions are also maintained in a hardware table arranged by range. Software also updates the shape factor table for changes in platform altitude, sea state and radar mode. The correlation of the sea clutter between successive PRIs is controlled by limiting the number of range cells replaced by hardware in the sea profile table during each PRI.

The gamma distributed random variables are obtained by searching a cumulative density function (CDF) table using a 32 bit uniformly distributed random value (from a 129 bit

LFSR counter) as the table key. The search is a seven iteration pipelined binary search of the 128 entry by 32 bit CDF table selected by the shape factor. The search locates the largest CDF table entry not exceeding the value of the 32 bit uniform random value. The resulting address is a seven bit random value with the probability distribution specified by the CDF corresponding to the shape factor for the current range cell. This technique permits an arbitrary probability distribution (a fact that is used to simplify the functional testing). The magnitude from a look-up table, delayed to align it with the gamma RV, scales the result to produce sea clutter profile values. The maximum required sea clutter replacement rate is 25 percent of the range cells per PRI. Since the pipelined binary search turns out a maximum of one new random value every seven clocks, the maximum achievable replacement rate is 14 percent. The CDF search logic and tables are duplicated to generate two values every 7 clocks, boosting the maximum replacement rate to 28 percent. The CDF tables are static, so the binary search FPGAs do not contain logic to provide host access to the tables. The host software programs the tables using a special FPGA configuration before the search design is loaded.

The random range generator produces a random sequence through the active range cells (those that are included in the PRI). The hardware discards range addresses outside of the active ranges before finding the gamma RV so that the maximum change rate is not degraded. The generator visits each cell in the active ranges before starting over with a new sequence. The hardware adjusts the length of the random sequence to the smallest power of two greater than the number of active range cells to minimize the number of discarded addresses. Each pass through the sequence is permuted so that the order the range cells are visited changes. Valid range addresses are queued in a FIFO buffer where they are kept until needed by the gamma random variable logic. Range generation is suspended when the FIFO gets full to prevent losing valid ranges. A counter is used to limit the number of cells changed per PRI. This provides a control over the degree of sea clutter correlation between successive PRIs.

The data provided by software for the shape factor and sea magnitude tables is compressed using run-length encoding in order to conserve MIX bus bandwidth. The clutter hardware performs the run length expansion when filling those tables. The table writes occur as needed between the scheduled table reads so that table updates do not hinder sea clutter generation.

The sea profile RAM is read out in range order during each PRI. The range ordered sea clutter and Gaussian noise profiles are combined using a root sum of squares function look-up table. This combined profile is passed on to the target module where it is combined with the receiver noise and then modulates a complex Gaussian noise source. The root-sum-of-squares function is implemented as a static look up table that is programmed by host software during module set-up (before the operational FPGA programs are loaded).

IV. BENEFITS OF USING FPGAS FOR PROCESSING

A. Algorithm Design Decoupled from Board design

The first significant benefit we saw from moving the design to a reconfigurable FPGA platform was that the board design became largely independent of the algorithm design.

The architecture for the FPP board was developed using the block diagrams for the proposed target and clutter modules as described earlier. This made sure that the number of memories per FPGA, number and size of FPGAs, and the interconnect structure was sufficient to host both designs without compromise. The architecture was then generalized by filling out the connections to make it symmetric. This helped to ensure the module could be used for other applications. (As it turns out, the architecture is better suited for many pipelined DSP applications than many of the commercially available FPGA processor modules).

Decoupling the algorithm from the board yields several significant benefits. First, the common board design is reused in multiple applications (by design). This eliminates the extra cost, risk and effort otherwise required for unique board designs. The general-purpose board can also be used for future and unrelated applications. A second benefit is the easy path for recovery from specification changes, algorithm tweaks and design errors. Since the algorithm is implemented entirely in reconfigurable logic, any changes are essentially code changes. That means there is no board rework required to recover from any design upsets. This permitted us to proceed with the board release much earlier than would have otherwise been possible.

Having the board design separated from the algorithm design also gave us the opportunity to fabricate and completely test the board before the FPGA designs for the algorithm were completed. This accelerated the design cycle in two ways. First, the board was not held up waiting for FPGA designs, and secondly the verified board became a tool for the FPGA development. By having a verified board available during the FPGA development, we were able to verify the FPGA designs on actual hardware in lieu of more intensive and time-consuming simulation.

Since the same board was used in multiple applications, the common elements in the design such as the FPGA interfaces to the host, the memory interfaces and I/O pin placements could be reused in many places in the design. This reuse allowed us to check these common parts once in a test design then use them with confidence in the operational designs. The reuse of key design elements also significantly reduced the design effort.

The on-board configuration memory holds 16 FPGA configurations, so all of the operational programs could be programmed into all modules regardless of the intended function. This avoided the nasty part numbering exercise that seems to happen all to frequently when the hardware is physically identical, but is programmed differently. There was enough room left in the configuration memory to also hold the

IFF transponder simulator design, which was eventually implemented on another copy of this board.

B. Highly Integrated Design

Using FPGAs as the design fabric also allowed us to achieve a greater degree of integration than was possible using other off-the-shelf components. The greater integration reduced the module count by two and eliminated many specialty parts.

The concept design used four devices (two STEL 1180's and a pair of multipliers) for each complex NCO to support the uncompressed model, plus a pair of Harris FIR chips for the compressed model. The target sort was to tentatively use content addressable memories to effect a fast sort. All of these were high dollar single source items, and all had peculiar interfaces required to set them up for operation. Instead, these functions were accomplished in FPGA logic, eliminating the specialty parts and the Rube Goldberg interfaces otherwise needed. Control of these functions was also vastly simplified. Designing these functions into the FPGA also opened them for customization to our needs. This allowed us to introduce a random start phase for the chirp NCO (something we previously had to work around) as well as to customize the word widths to our needs. It is worth noting that the performance of the FPGA designs met or exceeded the performance specifications for the special purpose parts in all cases, and that the FPGA solution has a lower total component cost.

A fixed function board requires all the logic for all modes of operation to be present whether the logic is currently used or not. In contrast, by using reconfigurable logic, mode specific designs can be loaded each time the RTG mode changes. This not only eliminates the logic for other modes, but also eliminates all the switching logic and configuration registers otherwise used to select the current function. We also took advantage of reconfiguration to pre-load static look up tables in memory, eliminating the logic otherwise required to read and write those tables. The idle logic is essentially cached in cheap EEPROM memory. By loading only the logic that is currently needed, the design is more compact, faster (less latency due to function select) and uses less power. The power consumption would have been more than 25% greater if all the functionality was resident in the FPGAs at once. This is significant, since each board dissipates about 30 Watts.

The logic used to create the target waveforms occupied nearly 3/4 of the available real-estate on the concept module design. By eliminating the interface logic, using reconfiguration, and tailoring the algorithms to FPGA implementation, this logic was reduced to two designs for one FPGA (Only one is resident in the FPGA at any given time). We were also able to reduce the load on software by adding additional multipliers and tables to implement the range attenuation (STC curve) and receiver noise in the hardware. This design shrinkage freed enough room to incorporate the PRI buffer function on the target generator module. With a little tailoring, the PRI buffer board design also collapsed into a single FPGA. Since there are two sets of modules in the RES, this integration reduced the module count by two. Combining the modules also reduced the inter-board cabling, thereby increasing reliability.

C. Debug and Integration Substantially Simplified

1) Board Test Independent of Algorithm

Using FPGAs and memory as the processor fabric also provided an outstanding opportunity for exhaustive diagnostics. A separate FPGA program was used to test the board function without having to worry about effects caused by the algorithms. Four copies of the test program simultaneously tested all of the memories at the clock rate using interleaved reads and writes and an LFSR generated test pattern. The test read address is the previous write address inverted. The read data is compared to a second LFSR with a reversed sequence and errors are accumulated to provide an error count. This test causes all of the address bits and many of the data bits to toggle on every clock cycle. This really is a worst case test pattern in terms of bit transitions, timing, data patterns, and noise. Passing this test at the clock rate assured us that there were no timing or signal quality gremlins waiting to bite us during algorithm checkout. Such a brutal test clearly is not possible using a more traditional host interface to the memory. Of course, the test program also gave us traditional host access so that we could check basic memory function before running the full clock rate test.

The test FPGA programs also allowed us to run test patterns through all of the interconnect wiring, bus switches and expansion ports. Like the memory tests, these tests run LFSR test patterns at the clock rate through each of the interconnect paths. Each line was driven in turn by each of the possible sources and sensed at all the destinations to ensure interconnect integrity. The connection and isolation of all the bus switch elements was checked. A loop-back cable connected between the expansion connectors allowed a full check of the expansion port registers and drivers too.

The ability to reconfigure the FPGAs to run high speed dynamic tests on the board gave us a 100% board checkout with very little test vector design effort. Without reconfigurability, such coverage would have been highly impractical if not impossible, very expensive, and would have required a monumental test vector design task. The largest benefit of using reconfiguration for board test is that it completely isolated the board test from the algorithm design. That meant that once the board was tested, the algorithm could be debugged with confidence in the electronic substrate.

2) Reconfiguration Speeds FPGA Design Tests

Algorithm and FPGA design also benefited from the reconfigurability of the FPGAs. The board design intentionally has identical connections for all four FPGAs. This makes the FPGA designs relocatable, allowing any FPGA design to be loaded into any of the FPGA devices. Relocating the FPGAs allows each FPGA design to be surrounded with

test designs. To test an FPGA, the host loads a test pattern into memory associated with a test design. That test pattern is then run through the FPGA at the design clock speed and the results are collected in the memory associated with the test program on the other side of the FPGA under test. After the test run is complete, the host can read and analyze the captured results at a leisurely pace. Similarly, combinations of FPGAs can be tested for function and performance. Where analysis was required at a sub-FPGA level, the FPGA design was modified to strip out logic following the desired pick-off. Testing the stripped down logic verified the portion of the circuit under test.

This access to individual FPGAs provides access to key points in the data stream without incurring a logic overhead in the design. More importantly, the ability to grab intermediate results with supplemental programs rather than with permanent access points in the design allowed debug at the design clock rate and eliminated costly test logic. Traditional testability in pipelined hardware allows the host to read the data stream at selected pick-off points. That usually requires the data pattern to be held static during the relatively slow host reads. A static data pattern can easily mask timing problems and data pattern sensitivities.

3) Hardware Verification Done Offline

Much of the software was developed on the actual RES system to avoid porting issues. Most of the software function was coded and checked without the hardware modules installed. The large number of people working on the project, and the limited number of systems put a high premium on system time, especially for exclusive use like that required for hardware debug. The host MIX bus interface is implemented in Lattice ISP programmable logic (selected for non-volatility and ability to be reprogrammed in-circuit). By reconfiguring these devices with a parallel port interface, we were able to perform the hardware debug off-line. The parallel port interface was specifically designed to appear exactly like the MIX interface when viewed from the FPGAs. This way, the FPGA application programs would run the same whether the host was the MIX baseboard in the system or a personal computer. By using a PC as the host, we were able to do the hardware development, debug and verification off-line (and for the most part off-site) using familiar tools, and without system availability issues.

A substantial part of the hardware debug effort consisted of writing C programs to exercise and analyze the module functions. As each function was verified, the test code was passed on to the software team to serve as interface examples. The astonishing result was that the hardware/software integration was completed ***within a week*** of when the software team first saw the boards!

V. CONCLUSIONS

The use of FPGAs as the hardware for a radar environment simulator provided custom hardware sufficient for a very complex simulation while also producing significant benefits. Benefits realized include reduced module count, flexibility to make design changes, simplified testing and integration, and a reusable module design.

REFERENCES

[1] Ward, K. D., Baker, C. J., and Watts, S., “Maritime surveillance radar. Part 2: Radar scattering from the ocean surface”, IEE Proceedings, Radar & Signal Processing, Vol. 137, Pt. F, No. 2, April 1990

[2] Stanford Telecommunications Inc., “STEL 1180 Data sheet”, Rev 2/25/94.

[3] Andraka, R. J., “A survey of CORDIC algorithms for FPGA based computers," Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Feb. 22-24, 1998, Monterey, CA. pp191-200