Designing a Generic, Software-Defined Multimode Radar Simulator For FPGAs Using Simulink® HDL Coder and Speedgoat Real-Time Hardware

Thomas Michael Simon Reynolds Thomas Woolford
Simbiant Pty. Ltd.
Adelaide, Australia
contact@simbiant.com.au

***Abstract*— We present a method for designing and realising Software-Defined Radar systems using Industry Standard software tools and COTS hardware. We describe how we built a generic multimode surveillance radar system in software, and how to interface the system to external RF systems by leveraging Field Programmable Gate Arrays (FPGAs). Using these techniques, we are able to increase simulation fidelity while reducing time to build and test our models.**

Keywords—RADAR, FPGA, Auto-Coding, Modelling and Simulation

I. INTRODUCTION

One of the biggest challenges in developing electronic countermeasures for deployment against a radar is "closing of the loop". There exist many examples of stimulators that can be used to test the detection, classification and technique selection capabilities of an electronic attack (EA) system (i.e. a Jammer). There are however limited options available to verify that a given technique is appropriate against the detected radar waveform in a dynamic environment. To investigate the dynamic behaviour of both the EA system and radar together it would be beneficial to have a multi-waveform, reactive surrogate that is easily modifiable, can represent the signal processing found in radar systems and can operate in a real-time environment, suitable to be interfaced with hardware.

Stimulators allow the EA developer to set-to-work, verify and test the signal processing of sub-systems, including the full system by generating one signal, through to a dense RF environment containing many interleaved pulse trains each with different pulse parameters (e.g. frequency, amplitude, pulse width and repetition rate). Sub-systems to be tested include (but not limited to);

- analogue RF components,
- digitisation,
- signal detection,
- search strategies,
- signal classification,
- signal identification

This is however still an open loop test, and does not supply responses from the target radar.

There also exist many tools to determine if the transmitted waveform matches what would be expected for the selected technique (e.g. spectrum analysers, oscilloscopes, etc.), but this doesn't test if the technique is effective. While some stimulators allow for limited closing of the loop, most of these are effects-based (e.g. when jamming is activated, change mode in a scripted way). Although this type of testing is essential for the development of an EA system, it doesn't test if the transmitted EA technique would have the desired effect against the signal processing of the targeted radar.

One method to test the effectiveness of an EA technique is to use modelling and simulation. This has many trade-offs, not least of which is determining to which fidelity level the signal processing is modelled. Modelling at the carrier or an Intermediate Frequency (IF) would require the simulation update rate to be in the order of MHz if not GHz. At this level the waveforms can be representative (carrier and modulation) and include many real-world effects (e.g. phase, frequency and amplitude changes within a pulse), but at the expense of computation time (e.g. milliseconds or even microseconds of simulation time can take days to run). Conversely a pulse descriptor word (PDW)-based model may run close to (or faster than) real-time but at the expense of how well the signal processing is represented. For example, a PDW model may not represent the complex harmonics and modulation within a pulse. Both these approaches have a place in determining the effect of a jamming technique: the waveform-level model (at RF or IF) can show how the jamming waveform affects the signal processing (raising the noise floor or generating false detections), whereas the PDW model can show the longer term effects (e.g. effects on tracking), but requires the model to be coded with how a jamming waveform would be interpreted by the signal processing. The PDW method requires detailed understanding and analysis of the radars signal processing and is consequently limited by it. Waveform-level method, on the contrary, allows novel techniques to be tested, and can contribute to system analysis. That is it can deal with the “Unknown unknowns” not just the “Known unknowns”.

The solution proposed in this paper is to use a waveform-level model operating at a sample rate that can represent a surrogate radar at an appropriate IF, with representative digital signal processing. As outlined above, a model of this complexity would be unlikely to run in real-time on a standard PC, thus a method of accelerating the simulation is required. There are now several options to accelerate a mathematical model, including compilation, distributed simulation, GPU acceleration, and Field Programmable Gate Arrays (FPGA) synthesis. Each has its pros and cons, and can have a significant impact on the run-time performance for different model types. For example a GPU solution can decrease the time required to process a large amount of bulk data where the data is all available at the same time, and each data point has the same processing performed on it (e.g. filtering an image). Whereas distributed simulation allows for a model to be broken up into sub-parts that can individually run in real-time, but with a limitation due to data transfer between sub-models. A FPGA solution is more optimised to streaming data, where the data arrives in small groups of simultaneous data (e.g. 1 sample per time step) and can be processed in a pipelined fashion.

An FPGA solution allows the modeller to build a system that runs in real-time, while combining a simple stimulator with waveform-level signal processing. The model: generates a representative transmit waveform (prior to up conversion to RF) that can be used to stimulate an EA system; contains the signal processing that is generally performed by a radar (where the received signal is a combination of target returns and jamming waveforms); and includes the radar dynamics as a response to the received signal, for example target detection and track.

Coding the radar signal processing can be performed using many different tools, from using VHDL with synthesis and implementation tools (e.g. Xilinx Vivado), through to graphical-based tools with auto-code functions to target the FPGA (e.g. MathWorks Simulink® with HDL Coder). For the radar control and tracking functions, system level languages with compiling tools (e.g. C++) through to graphical-based tools with auto-code functions (e.g. MathWorks Simulink® with Embedded Coder) can be used to target execution on a CPU.

The developer also needs to consider how the FPGA is interfaced with the CPU, such that initialization, state and output data can be transferred between them, along with controlling any peripherals attached to the FPGA or CPU. The developer can code their own drivers to perform the control and data transfer functions, or use an integrated solution (e.g. Speedgoat HDL coder workflow).

Finally as the radar model is to be used as a surrogate test system for an EA system, the FPGA and CPU solution must run in continuous (i.e. hard) real-time and not averaged real-time where the model can at times be slower than real time then faster to catch up (i.e. soft real-time). Thus a real-time operating environment is required with the ability to synchronise the execution of FPGA functionality with CPU functions (e.g. Speedgoat hardware).

II. Streaming Digital Signal Processing Design

Building a Radar model from specifications begins by blocking out the major elements of the processing chain. The processing chain will typically contain the following elements:

- Scheduling and Timing Control
- Waveform Generator & Arbitrary-IF up-sampler
- Arbitrary-IF Down-sampler & DDC
- Match-Filter Processing & Post-Processing
- Automatic Detection Unit and Clustering

Fig. 1 shows the processing pipeline for simple surveillance radar connected to an external Scene Generator via Analogue to Digital Converters (ADCs) and Digital to Analogue Converters (DACs), and performing basic signal processing.

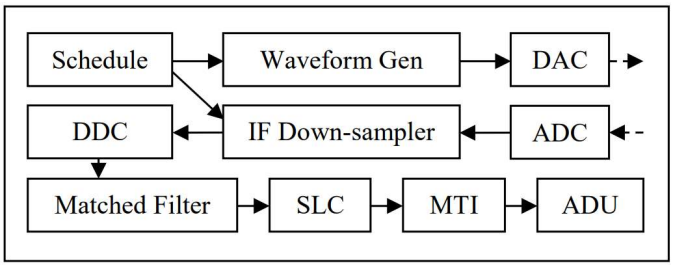

Fig. 1. Control and data flow within a generic radar processing chain

Due to the high timing precision required by the FPGA, waveform scheduling timing control is performed on-device and fed by a buffer of requested waveforms from the CPU. This means that there is a delay between processing the return from a dwell and scheduling a beam in response. The latency penalty is offset in communication reliability. Running in a low latency mode is possible, but more rigorous synchronization checking must be performed at the interface between the CPU and FPGA. HDL Coder also provides synchronization control between free-running (no sync, manual buffer control) and FPGA Interrupt driven (CPU interrupted by FPGA).

The ADC/DAC units used support sampling rates well above achievable clock rates of most FPGAs. We have implemented several FIR filters as time-stacked designs, which process batches of samples in parallel. IIR filters are ill suited to time-stacked designs, and are only used for scalar signals. These techniques are used to move between baseband and the arbitrary-IF interface with external hardware. Simulink® HDL Coder provides standardized blocks capable of high speed operation on the FPGA's Digital Signal Processing (DSP) hardware units, which reduce the time to produce designs that perform at high frequencies.

The Waveform Generator produces one or several baseband signals that have been scheduled for transmission by the radar. Simulink® HDL Coder provides several tools to make waveform synthesis simple, including parameterized Numerically Controlled Oscillators (NCOs) with adjustable dithering, direct phase control and low latency. Direct waveform playback from ROM can also be used. The generated waveforms are then up-sampled to a time-stacked format and mixed to the desired external Intermediate

Frequency, packed into a data-frame, and enqueued into an AXI4-Stream buffer for the DAC.

To recover the baseband signal on receive, an Arbitrary IF Down-Sampler first recovers the Quadrature channel through a time-stacked analytic filter, then mixes the signal to near-baseband. Using a time-stacked down-sampler, we can then extract just the portion of the incoming signal which would pass through the frontend filters at the desired virtual ADC sampling rate, clamping to system ADC resolution. This is then fed into the Digital Down-Converter which extracts the final baseband signal, reconstructing the quadrature component as necessary.

Drivers provided for Speedgoat Performance real-time target machines makes interfacing with DACs and ADCs a dropdown selection, indicating which in- and out-ports of the model correspond to each hardware unit. DMA communication to the host is similarly simple, allowing the direct recording of interesting waveforms to main system memory for later analysis.

Matched-Filter processing is the most variable part of the processing chain. We have developed several reconfigurable modules for common waveform types which can be chained together, or combined as needed to emulate a specific system. Modules are designed to collect them minimal amount of data before releasing processing results, minimizing on-chip block memory requirements. These include customized Fast Fourier Transform (FFT) architectures which trade-off between FPGA resource usage and latency, and can be swapped out as needed. Post-processing such as Side Lobe Cancellation (SLC), Side-Lobe Blanking (SLB), and Moving Target Indication (MTI) can also be performed at this step, modifying the Matched Filter output to extract more meaningful detections, suppressing clutter, or rejecting interferers.

The Automatic Detection Unit (ADU) takes streaming filtered data and uses a statistical model to determine which cells contain detections, or data of interest. Several methods are implemented from literature, and can be quickly tested against each other in simulated interference environments. Similarly, clustering can be performed in 1D or 2D, reducing detections to parameters suitable for tracking, latched and returned to the CPU at the next available transfer slot.

Simulink® HDL Coder provides support for this pluggable model in several ways. Using busses, we can latency match control and timing signals in bulk alongside the DSP pipeline. This simplifies the interface to blocks, and frees us from enumerating every signal which needs to be delay matched through that block. By making use of vectorized signals and For-Each subsystems, we can design modules independently of the number of channels they will process simultaneously. The number of channels then becomes a parameter to the model.

By using these tools, we are able to design libraries of generic processing blocks and templates to accelerate the implementation of new designs, and we can quickly incorporate new features into existing designs to gauge their effectiveness against previously effective techniques.

III. Scene Generation

A second FPGA card has been used to perform Scene Generation for the radar. The input to the Scene Generator is the full scale analogue output signals as created by the radar model. This signal is converted to the digital domain at a resolution depending on the hardware selected. The most basic form of the Scene Generator is a system which treats all returns as target like and possessing kinematic and reflectivity parameters that can be updated slower than one CPU clock period. This is because the scene generator is configured such that these required values are calculated CPU side and transmitted over to the FPGA to be used in waveform transformations. This divests chip usage from the FPGA.

The three key elements of this scene generator are: Delay, Doppler and Amplitude adjustment. The order in which these are performed presents as an example of design choices in the Simulink® HDL Coder onto FPGA workflow. It is desired that signal widths be kept at a minimum so as to require the least area per target. A single tapped delay buffer that records the pulse width of a signal can be used and the signals split for Doppler and Amplitude adjustment as the final step.

For the Doppler system, complex data is required to resolve whether the target is opening or closing. A software defined analytic filter can be used to recover the quadrature component of the input signal such that the complex mix can be performed. The positioning of the analytic filter has an effect on the hardware consumed on the FPGA. Placing the analytic filter after the delay line requires one analytic filter for each target, consuming adders and multipliers in the FPGA. Placing the analytic filter before the delay line consumes double the Block RAM in the device to store both the in-phase and quadrature signals. This choice will depend on a number of factors including the hardware available, number of targets emulated and pulse length received.

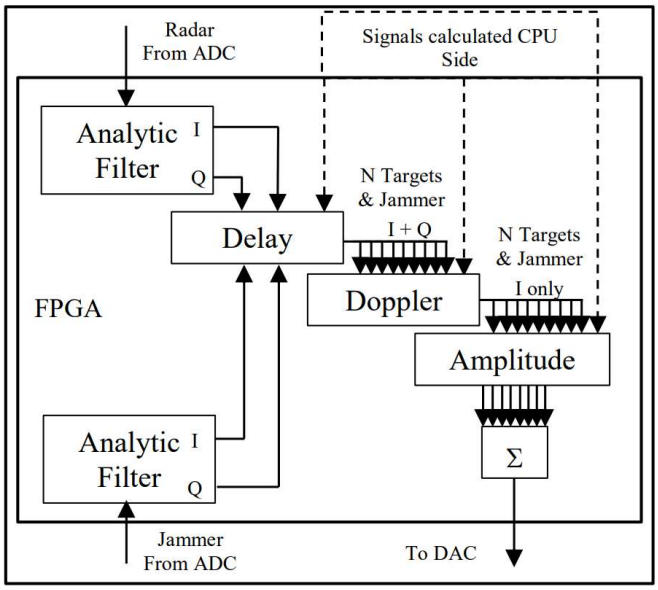

Fig. 2. Design of a Simple Radar Scene Generator to be used on an FPGA

Displaying the importance of configurable hardware and software in the designing of complete waveform-level software, *Fig. 2* shows an example design embodiment for a Scene Generator. It is desirable to possess a high configurability in the selected hardware such that more complex scene elements such as ground clutter could be added on the FPGA. If hardware area constraints do not permit, these advanced signals can be injected as analogue signals to the card to be added to the transformed radar signal. This same design methodology applies to the electronic countermeasures under analysis such as the jammer pictured in *Fig. 2*. In the above case the jammer scene parameters are entered, and the jammer waveform altered independently of the construction of target returns. The modular nature of adding additional signals allows for the simulation of a 'many vs many' engagement environment at the waveform level.

IV. Results

Implementation of the above architecture was done using an integrated solution by Speedgoat. Each Speedgoat Performance real-time target machine housed two IO342 Xilinx Kintex Ultrascale FPGA I/O modules. Each module is automatically programmable with Simulink models using HDL Coder, performing floating and fixed-point VHDL code generation from Simulink, synthesised by Vivado. Each IO342 I/O module provides up to eight 16-bit analogue inputs of type TI ADS54J60 and up to eight 16-bit analogue outputs of type TI DAC39J84 provided by Speedgoat IO342-63 (FMC120) FMCs, allowing for generation and consumption of gigasample data The PCI Express bus was used to transfer slow-time data from the CPU to the FPGAs.

The radar model contained capability for several common waveforms with tuneable parameters. This included:

- Linear Frequency Modulation with parameterised Pulse Repetition Intervals, Pulse widths and Bandwidths
- Unmodulated signals with parameterised chip width
- Binary Phase Shift Keying with parameterised phase encoding generation and chip widths.

The Scene Generation model operated with the following restrictions due to the hardware used:

- Maximum number of targets limited to 16
- Maximum input pulse length 400 us
- Maximum delay 1 PRI (Returns from signals that are not from the previous transmission are not emitted)

These parameters required a full HDL build to be modified. The radar had internal scene generation functionality that could be used in the system test for the radar. Errors due to hardware / external scene generation were isolated and eliminated due to the internal nature of these tests.

Test targets were received and resolved from the Scene Generator at expected ranges between the minimum and maximum for each radar waveform as defined by pulse widths and pulse repetition intervals.

Tests were also conducted to test susceptibility of the radar to jamming waveforms. Jamming signals were produced on a second Speedgoat device and injected into the Scene Generator. Using a CW LFM Jammer of frequency slope equal to the radar's when in LFM mode, clear degradation in performance at the detect level was experienced. The targets were not logged at the detect level and the buffer was filled with false targets. This displays the effectiveness of the devised solution in its intended use case.

Using the above configuration, it was possible to begin with a Simulink® model and finish with a real-time solution within 8 hours. The radar took less than 5 hours to build and was more susceptible to failed builds due to timing failure of place and route. This required the further constraining of model elements to areas on the FPGA to assist in the synthesis process. The Scene Generator took less than 3 hours to build and often did so without consuming a large proportion of the Kintex Ultrascale FPGA's resources. This suggests the capability of the Scene Generator could be improved without great difficulty. The current time for which the complete model builds has improved incrementally with MATLAB version, Speedgoat Support package version, and efficiency improvements of user created blocks. Previously the total build time of the model exceeded 16 hours. Leveraging the incremental improvement of the Simulink® auto-coder and targeted FPGAs displays the advantage of the integrated hardware and workflow adopted.

V. Conclusion

We have implemented and tested a fully-parameterised radar prototype using off-the-shelf tools that fulfils our original objectives of high-fidelity real-time execution. By combining the visual system construction language Simulink® with new tooling for FPGA-based Digital Signal Processing, we can build libraries of components that can be re-combined and re-used to build new radar architectures quickly and cheaply.

We believe this is an exciting new avenue to explore the interaction of multiple systems in a shared RF environment, and to test system capabilities against known RF hardware and waveforms. Future work will focus on expanding the scope of waveform types and processing algorithms that we support, and in refining the performance of our current implementations.

VI. Acknowledgment

Simbiant thanks DST Group Edinburgh, Speedgoat and MathWorks for their generous support and cooperation.