10.3 A General Purpose FPGA Based Radar Controller and Signal Processor

Charles Martin*, Eric Loew, Chris Burghart
National Center for Atmospheric Research, Boulder, Colorado

1 Introduction

Radar implementations began as fully analog systems. Digital technologies have held substantial roles for many years, but until the last two decades the major digital functionality largely began at the analog to digital and digital to analog conversions that occurred at baseband. Signal down- and up-conversion, as well as specialized pulse shaping, were performed with discrete hardware modules. As analog converters became capable of operating at IF sample rates, it became possible to migrate the IF processing into the digital realm. Typically this was performed by custom circuit boards designed around specialized components coupled with onboard digital signal processors. This hardware and embedded software solution then performed the up- and down-conversion and other signal processing functions, and allowed the communication with the host computer to occur at baseband sample rates.

In the past decade, new capabilities have emerged in the digital arena, due to the increased capabilities of the Field Programmable Gate Array (FPGA). This device provides a foundation for building general-purpose hardware that can be adapted to the signal processing needs of many types of radar (or other) systems. Because the FPGA is a re-configurable, generic, and high-performance computing device, it can serve as the central component in a general-purpose control and signal processor subsystem. Indeed, there are now commercial transceivers available that can be utilized in just this manner. Radar system designers can configure a purchased transceiver card, rather than having to develop complex and specialized hardware.

The purpose of this paper is to summarize one such project, and provide guidance for those who are exploring the idea of developing similar capabilities. A brief introduction to the FPGA is given first, followed by a description of the processor architecture. Examples of some diverse applications of this system are presented, followed by a discussion of benefits and pitfalls.

*Corresponding author address: Charles Martin, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307; email:martinc@ucar.edu

2 The Field Programmable Gate Array

The FPGA is simply an integrated circuit with an extremely large number of generic digital logic cells. These cells can be dynamically connected to create building blocks capable of basic digital functions. These components are then combined to produce more sophisticated logic, and so on until very complex functionality is realized. For example, devices with the complexity of complete CPUs, DMA memory controllers, and PCI bus interface controllers can be synthesized for modern FPGAs. In fact a single FPGA will often be performing all of these functions and more at once. The FPGA is also well suited to implementing pipelined computations that can be highly parallelized, such as filtering, frequency conversion, and the fast Fourier transform. These allow us to implement much of the functionality, originally located in analog or discrete digital components, in the re-configurable digital environment of the FPGA.

The FPGA device will have digital I/O connections for bringing signals into and out of the chip. The number of digital I/O lines can number in the hundreds for the larger devices. Processing within the FPGA is driven by numerous clock signals, which gate the digital data through the configured cells. Clock management is one of the more challenging aspects of firmware design, when dealing with mixed frequency applications. This is common for designs that are connecting signals that are derived from IF rates to computer buses, memories, etc. Fortunately the FPGA development tools provide capabilities for correctly managing the intersection of clock domains.

3 The FPGA Workflow

The FPGA configuration is commonly specified with a hardware specification language such as the VHSIC Hardware Description Language (VHDL). A sample of VHDL code is presented in Figure 1. The developer creates text descriptions that define the connections and operations of entities such as signals, registers, and busses. Basic logic and numeric operations, for example latching, addition and shifting, are

```
--! Determine if we are on the even or odd pulse
--! by counting the gates.
even_odd: process (CLK, RST, g, even)
   begin
      if (RST = '1') then
         even <= '1';
         g  <=  std_logic_vector(conv_signed(0,  16));

      elsif (rising_edge(CLK)) then
         if (VALID_IN = '1') then
            if (g = GATES-1) then
               even <= not even;
               g   <=   std_logic_vector(conv_signed(0,
16));
            else
               g <= g + 1;
            end if;
         end if;
```

Figure 1. Example VHDL source code.

defined in the language. Sophisticated computational components are constructed from these building blocks. These components are then "wired" together to implement a complete control and signal processing system.

The complete FPGA configuration will be defined in a collection of VHDL text files that can be modified using any source code text editor. However, the editing activity is usually performed within an integrated development environment. The Xilinx Integrated System Environment (ISE) is a very well known development package. It provides an integrated collection of tools that manage the complete FPGA workflow. The typical sequence of development is as follows:

- Edit the VHDL file(s)
- Test via simulation
- Synthesize a bitstream
- Download the bitstream to the FPGA
- Test and debug
- Return to editing

Some of these steps can be quite lengthy, and so one cycle through the workflow can be very time consuming.

Simulation is essential to efficient FPGA development. A test vector of simulated inputs are created and wired to the component, and the clock input is driven in order to run the simulation. The outputs are verified for correct behavior. Higher-level modules comprised of multiple components are simulated and tested in the same manner. Tools for facilitating simulation are provided by the development environment. The simulations are accurate enough that in general if the VHDL simulates properly, and the synthesis step is free of errors, the generated firmware will execute properly on the FPGA. Crafting and successfully passing simulation test benches are critical to creating reliable FPGA firmware designs.

The synthesis step is where the VHDL description is converted into a specification for the digital cell connections on the FPGA. There are several sub-steps in this process. First the language description is translated into a network of logical register specifications. The register specification is then mapped onto the basic component types available on the particular FPGA being used. Finally, this logical network must be assigned to the actual logic cells within the physical FPGA. This last activity is known as "place and route", and it is a significant computational challenge. The place and route task is both combinatorial and heuristic, and there is no guarantee that the tool will be able to reach a solution that will meet the timing constraints of the design. The development tools provide facilities that allow the human to provide some guidance, such as suggesting logical locations to place major components of the design. The place and route algorithm is iterative, as the software attempts to converge on a solution that meets the timing requirements. It is not uncommon for the place and route task to require 12 hours or more on a capable workstation. For this project, place and route runs in about 20 minutes.

The synthesis part of the workflow can cause the FPGA workflow to be tedious. This is another reason why simulation can greatly enhance the development cycle.

The product of synthesis is the "bitstream", which is the lowest level specification of the FPGA connections. The bitstream must now be made available to the FPGA. It can be downloaded directly to the FPGA, but it is more common to "burn" it into a non-volatile memory such as an EEPROM, which will later be loaded into the volatile FPGA. In either case, the download can be performed via a JTAG interface, which provides a direct hardware connection to the FPGA device. If the FPGA is part of a card located on a host computer's backplane, there will often be a mechanism for loading the FPGA directly across the host bus. This eliminates the need for specialized hardware to support the JTAG capability. In some cases only the volatile FPGA load can be performed via the host backplane, and the JTAG capability is still required for programming the non-volatile storage.

Once the FPGA has been configured with the bitstream, system testing and debugging can be performed. If FPGA problems are detected, debugging can be initiated. Usually this must be

done indirectly, such as by altering the system configuration in order to expose and diagnose the FPGA behavior. Xilinx also provides a very powerful method (ChipScope) for inserting firmware probes into the VHDL code so that the logic behavior can be tracked directly within the operating FPGA.

Another powerful technique exists for creating firmware components. A processing algorithm can be written in a high-level language, such as Matlab. This code is then compiled to produce a firmware component that can be integrated with the VHDL code. Matlab provides a complete toolbox for crafting signal processing elements such as filters, fast Fourier transforms, etc. The full functionality of Matlab is available to investigate and characterize the performance of these components.

4 Transceiver Cards

The FPGA provides the computational foundation for a radar controller and signal processor, but it must be integrated with other hardware. A transceiver card is required which can provide receive, transmit and control signal functions. Commercial vendors provide products that provide this general-purpose functionality. The FPGA is typically combined with analog-to-digital and digital-to-analog converters, local memory, digital I/O lines and bus interfaces. The Pentek Model 7142 (Figure 2) was employed for this project. As is common with these products, this compact card is based on the Portable Mezzanine Card (PMC) format. A wide variety of carrier cards are available to host the PMC in common computer backplanes.

The hardware vendor typically offers a board support package, which provides drivers and application libraries for the host system, and firmware source code for the FPGA. The first two provide facilities to access the card on various operating systems. The firmware package can be a version of VHDL code that will run on the FPGA and provide baseline functionality, so that the card can be used in some cases without any VHDL development. The end user can customize the supplied VHDL for their specific purposes.

It should be noted that the quality of the firmware package will have a significant impact on the success of the development. It should be logically organized and well documented, both in the VHDL files and with other documents. Schematic type drawings are invaluable. Ideally it will come with a complete project specification for a development environment such as ISE. The best packages will contain explicit user blocks embedded in the data flow, to facilitate insertion of custom processing. The board support for the Pentek card was found to be top notch in this regard.

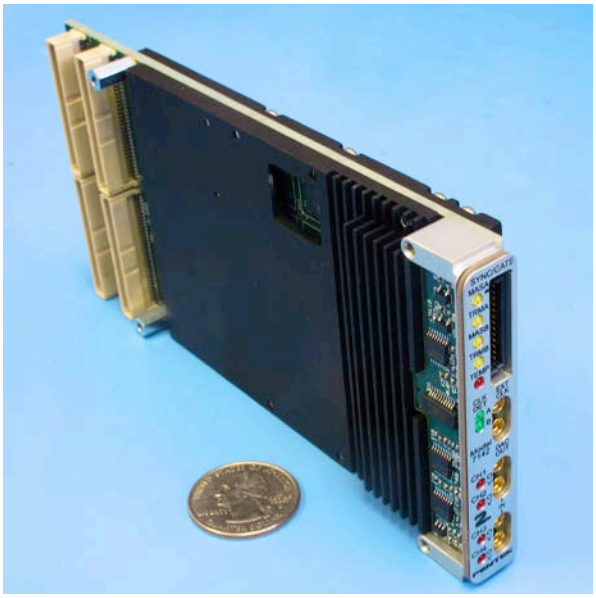

Figure 2. The Pentek 7142 FPGA transceiver. The FPGA integrated circuit is enclosed within the heat sink.

5 Radar Implementations

High-level diagrams depicting radar systems both with and without an FPGA based transceiver card are shown in Figure 3. The transceiver card replaces a large number of discrete components. This consolidation has several benefits. The design, construction and testing of some expensive analog sections are completely eliminated. Because the frequency conversion and filtering occur in the digital domain, the processing responses are completely matched and characterized without any measurement procedures. Most important though, is that all of these functions can be tailored for specific applications, with no additional hardware modification.

6 The Software-Defined Digital Down Converter (SD3C)

The SD3C is a framework that implements the radar signal processing and control functions. It is built on: (a) firmware operating on a Pentek 7142 transceiver card, and (b) software support libraries for the host computer system. These two subsystems interact via the computer bus that the card is attached to.

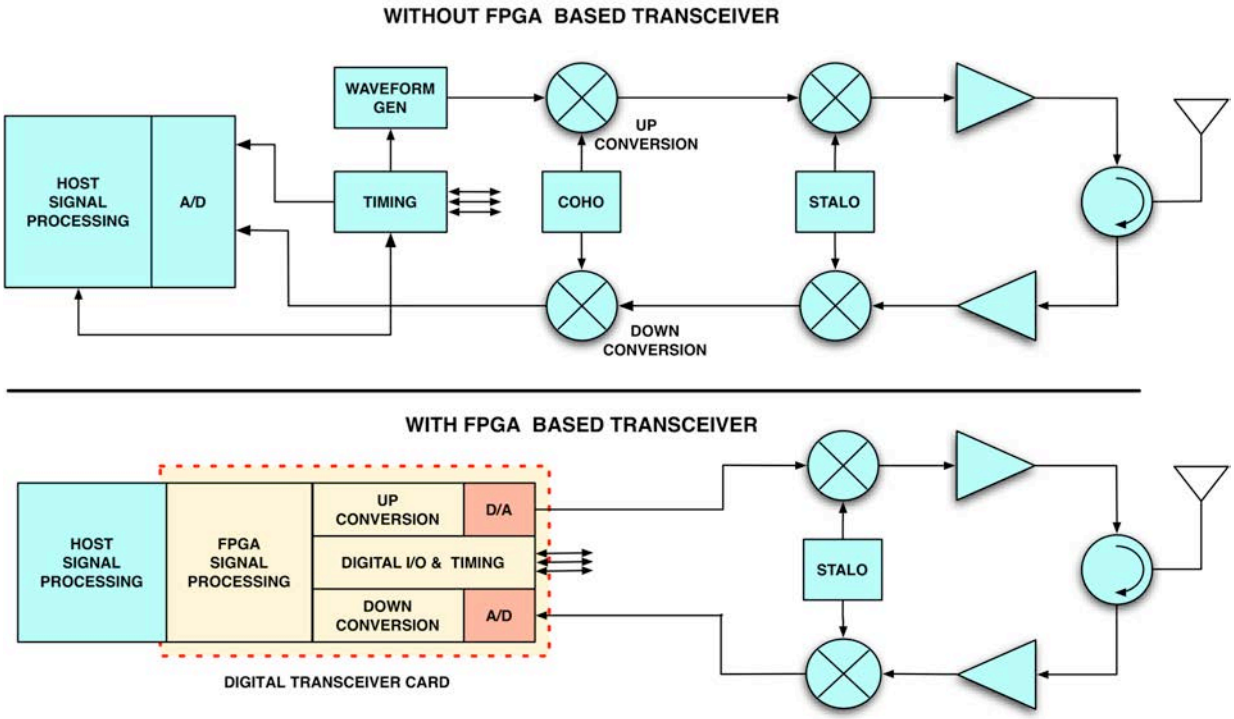

Figure 2. Schematics of a radar system with and without an FPGA transceiver. The yellow boxes denote functions that are performed by the FPGA firmware.

The SD3C implementation is described here in terms of the Pentek card that it currently runs on, but it should be noted that a very similar architecture could be realized using many of the commercially available transceiver cards available. Much of the firmware could be reused with other vendor hardware.

6.1 SD3C Firmware

Figure 4 presents a high level diagram of the SD3C signal-processing pipeline.

The SD3C firmware builds upon the Pentek "GateFlow" infrastructure. GateFlow provides a reference firmware design that is capable of running the transceiver card with basic A/D sampling, D/A output and digital I/O. The Pentek card is quite sophisticated, providing flexible capabilities for onboard memory capture, data routing and device configuration.

GateFlow provides specific "user blocks" that intercept the data flow through the FPGA, and into which the user can insert custom processing. The user blocks meet some but not all of the SD3C needs. The down conversion, matched filtering, pulse tagging, coherent integration and complimentary code processing are implemented in VHDL components that are instantiated in the user blocks.

GateFlow is oriented towards communications processing, and therefore does not inherently provide the functions needed for pulsed systems. This required other areas of GateFlow to be modified so that the down-converted data signals could be sampled on the pulse repetition frequency (PRF) boundaries. Similarly, the transmit pulse is generated by feeding baseband values to the up-converter, and GateFlow was modified to also allow this to be synchronized with the PRF.

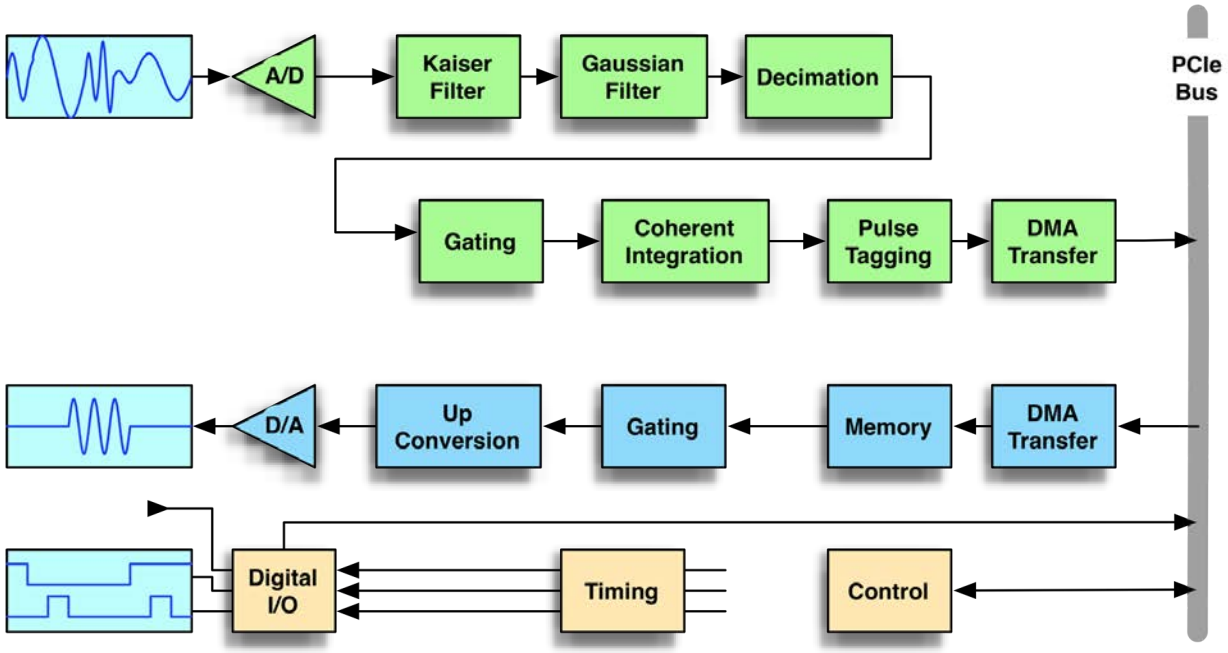

Figure 3. The SD3C firmware architecture

A programmable VHDL timer component was created in order to generate timing signals. Multiple copies of the timer are instantiated for the required timing signals, such as the PRF, blanker, and amplifier power control. These signals can be used within the FPGA to gate the signal processing, and they are routed to the FPGA I/O pins, where they can be brought off board for connection to other hardware. The current design provides eight independent timers. If additional signals are required, it is a simple matter to instantiate additional timers.

Many of the custom functions within the SD3C are configurable. Typical parameters include filter coefficients, timer periods, integration intervals, etc. GateFlow has a control register module that allows host software to write and read configuration registers for the existing GateFlow functions. This module was extended to allow the same configuration for SD3C functions.

Table 1 summarizes the SD3C custom VHDL functions that are incorporated into the GateFlow foundation.

Block	Function
Timer	Implements a gang of timers for timing signal generation. All timers are started by a common pulse, but are free running after that.
Control	Provides the interface for reading and writing SD3C configuration and control registers.
Down Converter	Combines an f_s/n down converter, a Kaiser filter and a Gaussian filter.
Pulse Tagging	Prepends an incrementing PRF pulse number, I/Q identifier, format identifier, and channel number to each received pulse.
Coherent Integration	Sums the samples for each range gate, for I and Q, for each channel. If complimentary coding is enabled, the sums are further segregated by even and odd pulse.

Table 1. SD3C VHDL components.

6.2 SD3C Host Software

The transceiver card and its firmware provide the hardware component for the SD3C. A software infrastructure implements a high level capability for the host computer to interact with the card.

As is common with most commercial transceiver cards, the vendor markets a low-level software interface that controls the basic card functions. In this case, Pentek provides the “ReadyFlow” package to support the P7142. ReadyFlow allows the user to configure and capture data from direct memory access (DMA) bus transfers, and to configure other functions in the FPGA.

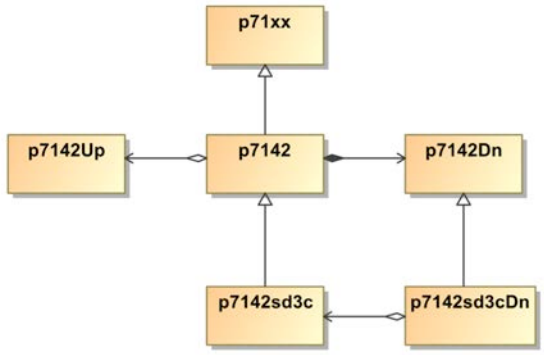

Figure 4. Simplified C++ SD3C class diagram.

The SD3C package builds upon ReadyFlow to create a programming interface that represents the characteristics of a radar processor, and manages the custom capabilities of the SD3C firmware. This takes the form of a C++ class library, whose class hierarchy is shown in Figure 5. The functions of the primary classes are described in Table 2. These three classes are all that is required in order to operate the transceiver as a radar system. The typical program sequence is as follows:

- Create a p7142sdc3 object, representing a Pentek P7142 card.
- Create a p7142Up up-converter object, for configuring the transmit pulse generation.
- Create p7142sd3cDn down-converter objects, one per channel.
- Configure the timers on the p7142sdc3.
- Issue a start command to the p7142sdc3.
- Loop on read() calls to each of the p7142sd3cDn objects, ingesting the baseband received data.

Because the SD3C prepends identifying tags to each received pulse, the p7142sd3cDn class is able to detect if data are dropped during the DMA transfer between the transceiver and the host system. This can become an issue on a heavily loaded host with a large data bandwidth. The tags also allow accurate time stamps to be applied to the pulses, which would otherwise be ambiguous due to data transmission lags through the system.

Using the SD3C C++ class library, a complete radar system can be developed. A typical host software architecture is shown in Figure 6. The digital transceiver (DRX) process controls the transceiver card, reads baseband data from it, and publishes the real-time stream to multiple subscribers. The subscribers are visualization, archive, and derived product processes. Some of these processes may publish results for other downstream subscribers. Segregating the functions using a publish/subscribe model allows the processing to be distributed among multiple hosts on the network, as needed. Figure 7 illustrates a real-time visualization application.

Class	Function
p7142sd3c	Overall transceiver control
P7142sd3cDn	Down-conversion channels
P7142Up	Up-conversion channels

Table 2. SD3C main C++ classes.

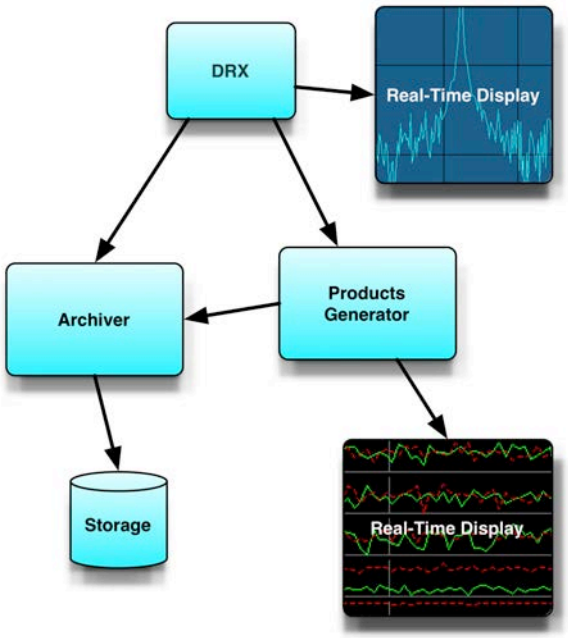

Figure 6. A typical host software architecture.

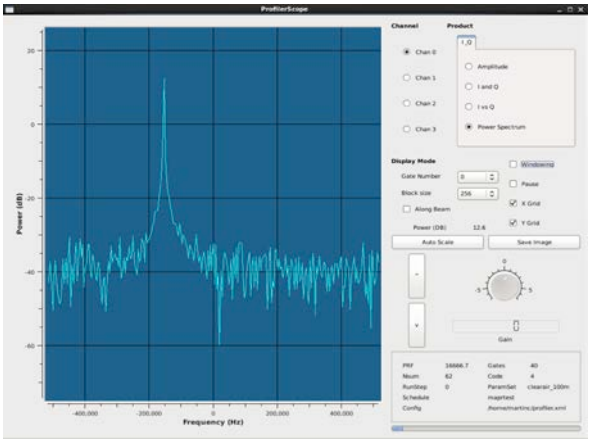

Figure 7. A software spectrum analyzer and oscilloscope.

	449 MHz Wind Profiler	W-Band Cloud Radar	K_a-Band Microphysics Radar
Application	Boundary layer dynamics	Cloud microphysics	Cloud microphysics and water vapor retrieval (with S-band radar)
Gate spacing	150m	37m	75 m
Range	7 km	15 km	7 km
PRF	40 kHz	10 kHz	1 kHz
Peak Power	3 kW	1.5 kW	35 kW

Table 3. SD3C radar applications.

Experience has also shown that separating specific functions into independent processes leads to a system which is simpler to construct, more robust, and easier to troubleshoot. It is important that the data distribution scheme allows the connections between two processes to arbitrarily come and go, without negatively impacting either process. The NCAR 449 MHz wind profiler has successfully employed this architecture, using the Open Data Distribution Service (OpenDDS) as the data transport mechanism.

7 Example SD3C Applications

The SD3C is currently employed in three operational radars, whose characteristics are summarized in Table 3. These systems cover a wide range of capabilities.

Images of the FPGA transceiver host, the antennas, and sample observations are presented in figures 8 through 16 for each system.

8 Conclusions

FPGA development has usually been the domain of specialized digital hardware engineering teams. In contrast, the SD3C project was executed by a very small group of RF system and software engineers, who had to learn the technology “on the fly”. The learning curve is challenging, and required gaining expertise in VHDL concepts and FPGA system design. The development workflow is intricate and time consuming, and the support tools are complicated and often non-intuitive. Debugging and testing can be very difficult. Based on our experience, the following recommendations can be made to engineers just starting with this technology:

- Gain confidence by starting with the basics of VHDL, implementing very simple components that can quickly transit the workflow, and that can be tested easily.
- Make the design as modular as possible, and reuse VHDL code wherever possible.
- Create and run simulation test benches for all components.
- Make full use of embedded VHDL probes (e.g. ChipScope) to diagnose problems right on the FPGA chip.

Use of FPGA technology transforms many aspects of radar system development into a major software activity. This work benefits greatly by employing standard software engineering practices. Designing from a software architecture perspective, using tools such as integrated development environments, source code revision control and bug tracking, and creating embedded documentation will all greatly enhance the project productivity.

The SD3C project has demonstrated that an FPGA solution, based on commercial hardware, brings many benefits to system developments such as those described here. The FPGA is capable of very high signal processing performance, thus mitigating large data bandwidths and host CPU loads. The technology facilitates very flexible applications: the same hardware can be customized to meet quite different requirements simply by loading the application specific firmware. Total system costs are reduced by leveraging the use of a single FPGA card among several systems, and by the consolidation of functions from many discrete hardware components onto a single card.

Figure 8. The spaced antenna array.

Figure 9. The host workstation. The FPGA transceiver is the card with the three attached coax cables.

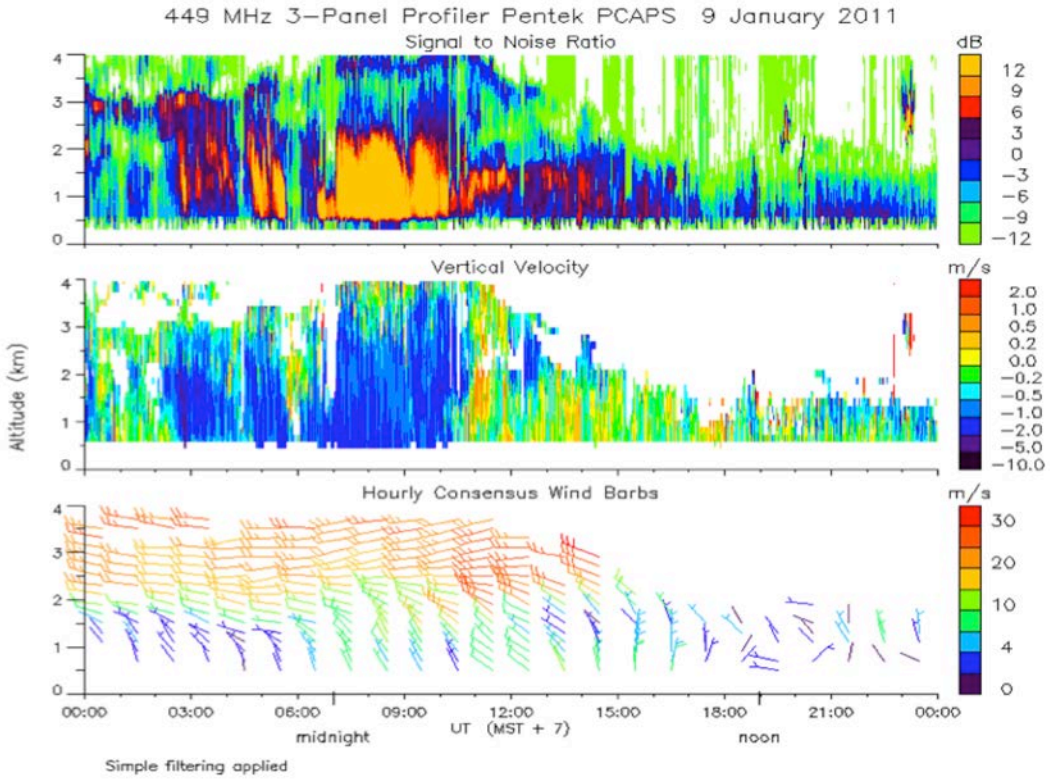

Figure 10. Time-height cross sections of signal-to-noise, vertical velocity, and horizontal winds.

Figure 11. The transmit and receive antennas, with a horizontal reflector for ground testing.

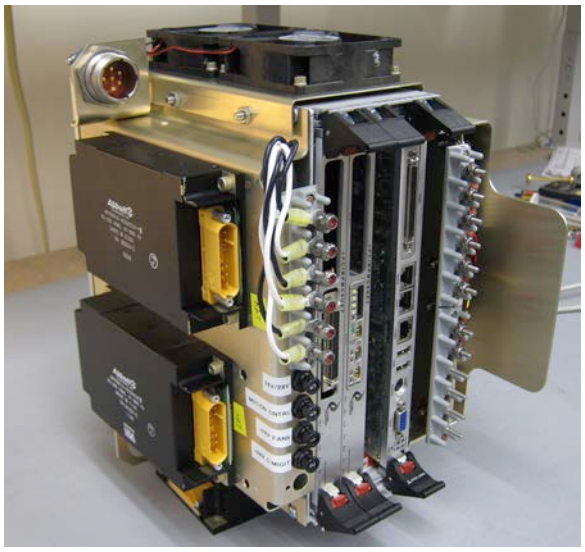

Figure 12. The host computer chassis. The FPGA transceiver is mounted in the carrier card; second slot from the left.

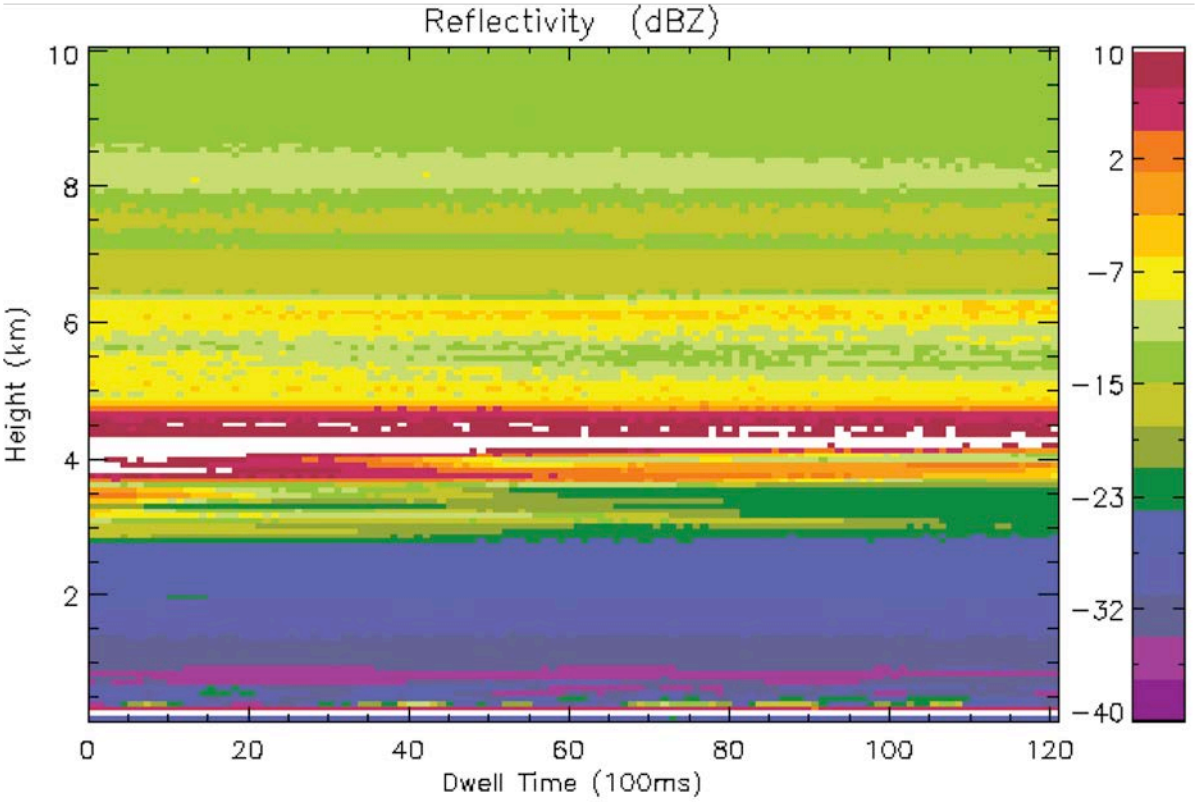

Figure 13. A time-height cross section of reflectivity.

Figure 6. The large S-Band antenna carrying the smaller Ka-band antenna on the lower right.

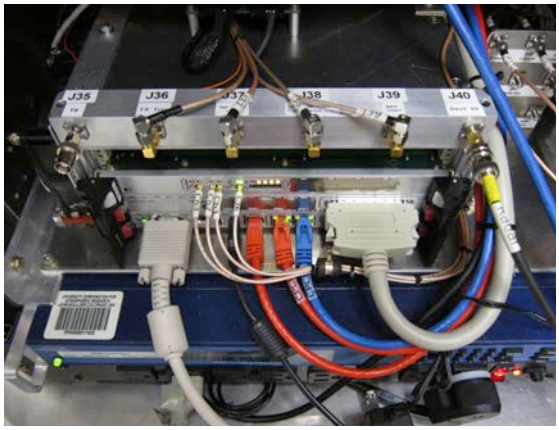

Figure 5. The host computer chassis. The FPGA transceiver is mounted in the compact PCI carrier below the four larger coax cables.

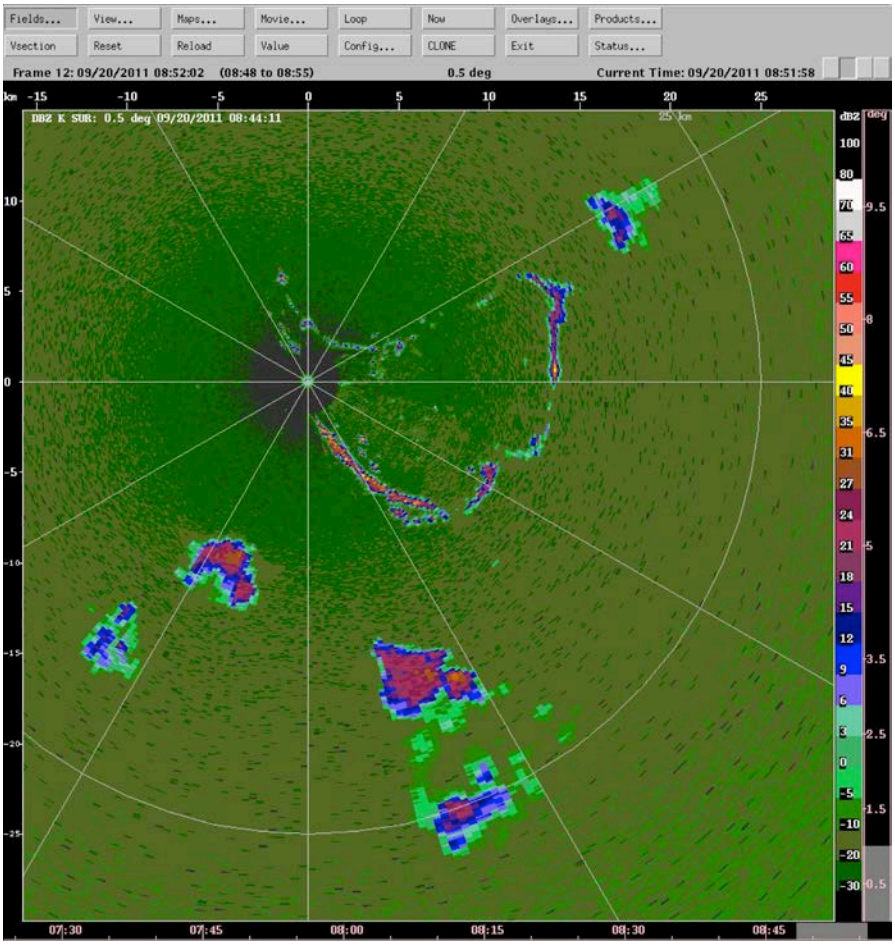

Figure 7. A reflectivity PPI display from the K_a-band radar.