Chapter 8. Rapid Prototyping for WLANs

Dennis McCain

IN THIS CHAPTER

In the interest of quicker time-to-market, it is important to be able to rapidly prototype new technologies enabling the technology transfer necessary for product development. The rapid prototyping methodology reduces the risk involved when deciding whether to implement new algorithms for a product and allows the validation of research ideas in hardware long before production commitments. It is a relatively low-cost way of determining the viability of products prior to spending millions of dollars on full-scale production. Rapid prototyping also gives a company a lot of leverage in setting the standards for new technologies; it is difficult to argue with a working prototype. For these reasons, rapid prototyping is becoming a standard among leading technology companies trying to keep up with a rapidly changing marketplace.

This chapter is divided into three sections. In the first section, an overview of system modeling is presented along with a rapid prototype design flow which includes a description of the hardware and software tools involved with its implementation. In the second section, a short overview of good digital design practices is presented describing some of the implementation issues with taking a system design down to silicon and getting it to run real-time. In the last section, a case study is presented describing a rapid prototype design flow applied to the prototyping of a real-time implementation of an IEEE802.11a WLAN (Wireless Local Area Network) baseband radio system. The prototyping methodology presented in this chapter was developed in conjunction with a research project focused on advanced IEEE 802.11a baseband algorithms; these algorithms, which were validated in the prototype described in the case study served as the reference design for productization in an ASIC (Application Specific Integrated Circuit).

Introduction to Rapid Prototype Design

Rapid prototyping is any design methodology whereby a system-level design specified in a high-level description language like C is quickly translated to a hardware implementation. The essential aim of rapid prototyping is to quickly produce a working system without going through the traditional time-consuming process of separately defining a firm system specification and then handing it over to a design team to implement. A common problem in translating a system-level design to a hardware implementation is bridging the gap between system engineering and hardware design. System engineering traditionally specifies a system in the form of documentation, pseudo-code, and so on and defines the partition between hardware and software. These specifications are then handed off to the respective hardware design groups which translate the system-level specification to an HDL (Hardware Description Language) such as VHDL (Very high-speed integrated circuit Hardware Description Language) [11] or Verilog. The advantage of using an HDL like VHDL or Verilog is that it allows hardware designers to describe the architecture of their design without worrying about the transistor-level implementation. This requirement to redescribe the high-level system design in a hardware description language creates a rift between system engineers and hardware engineers since this is done in a different development environment and generally only the hardware engineers understand their HDL implementation. As well, the translation to HDL inherently creates the possibility for errors and inconsistencies in the design as hardware designers are forced to reverse engineer the high-level system design.

To bridge the gap between system engineering and hardware design, a lot of time and effort is expended to write the specifications as clearly as possible and resolve any inconsistencies in the design. In large projects with deliverables in the 12–24 month timeframe, this system flow has been used in many companies; however, in a rapid prototype design flow with deliverables in the 6–12 month timeframe, this

process is not the most efficient. With rapid prototyping, it is critical to streamline this process so that system engineering comes closer to specifying the actual hardware implementation.

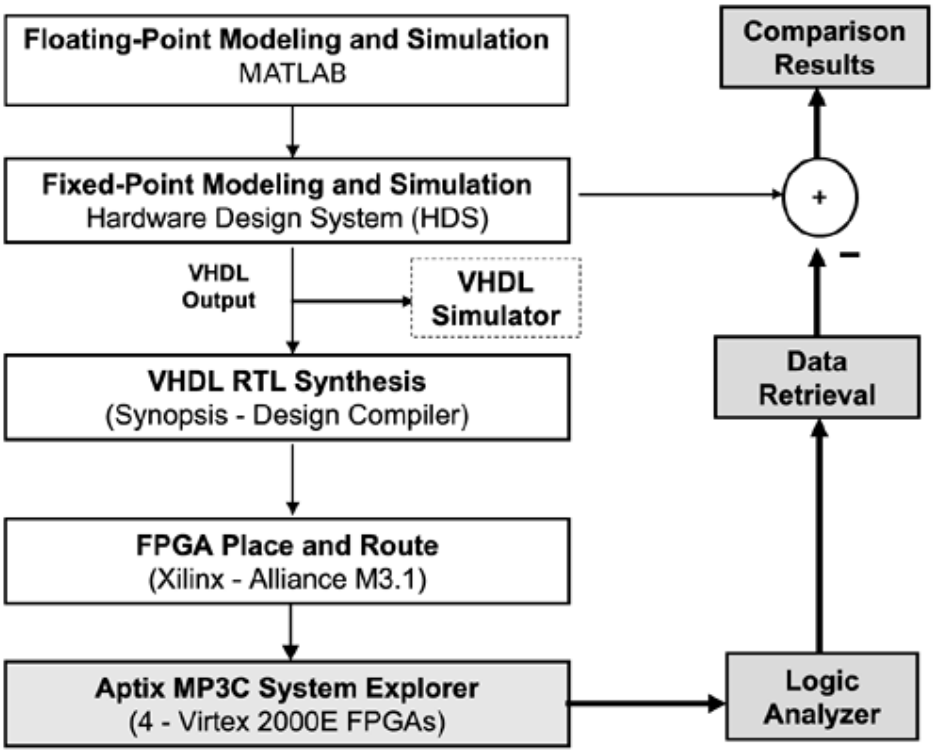
In an effort to bring system engineering and the hardware/software design closer, several EDA (Electronic Design Automation) companies have developed tools that make the translation from system algorithm to silicon design flow, creating the so-called system-on-chip (SoC), as seamless as possible. A number of leading system houses are supporting the Open SystemC Initiative (OSCI) [8], which uses a standardized subset of the C++ programming language to model hardware at the system level, behavioral level, and register transfer level (RTL). SystemC [9] provides a C++ based platform, which gives system engineers and hardware engineers a common development environment to model hardware at a high level of abstraction thereby bridging the communication gap between the two. This SystemC code-based hardware model can then be translated to HDL and then synthesized to a target hardware architecture. Because C/C++ lacks the syntax to adequately describe hardware, EDA companies have traditionally developed their own proprietary extensions to C/C++, which has fragmented the market for system level design tools. The OSCI seeks to make SystemC a standard for system-level design much like VHDL and Verilog are standards for hardware design. Once standardized, SystemC is an approach that can make rapid prototyping of new technologies and faster time-to-market a reality.

Another approach for system-to-silicon design that makes rapid prototyping possible is a graphical HDL entry design flow which allows designers to capture their high-level design in schematic-like block diagrams. Like the SystemC approach, this hierarchical, block diagram approach also attempts to bridge the gap between system engineering and hardware design by providing system and hardware engineers a common development environment in which to communicate. This common environment enables simple system model comparison and validation and ensures a more accurate hardware implementation. In the graphical entry approach, behavioral and architectural system design blocks are wired together graphically in a CAD (Computer Aided Design) environment and simulated for verification. Following the verification of the design, synthesizable RTL can then be generated to target a specific hardware architecture like an ASIC or FPGA (Field Programmable Gate Array). This design flow allows a fairly small group of system engineers to build complex systems in hardware without a thorough knowledge of hardware design. This design flow essentially allows the designers to focus on the design as opposed to the target hardware implementation. This rapid prototype design flow will be the primary focus for this chapter concluding with a case study of a project that successfully used the design flow in developing a real-time IEEE 802.11a baseband radio.

An Example of a Rapid Prototype Design Flow

An example of the rapid prototype design flow that is based on a graphical entry RTL tool called HDS (Hardware Design System) is shown in Figure 8.1. The design flow shown in the figure is the design flow used in the rapid prototype case study presented later in this chapter. The design flow begins with floating-point modeling and simulation in MATLAB (www.mathworks.com), which is a tool commonly used in the research and development of communication systems. MATLAB allows the rapid verification of communication algorithms before any fixed-point modeling is done where BER (Bit Error Rate) and PER (Packet Error Rate) curves are generated and compared to ideal system performance.

Figure 8.1. Example of a rapid prototype design flow.



Once algorithms have been optimized in the floating-point environment using MATLAB, the next step in the design flow is to do the fixed-point modeling of the floating-point algorithms. As described earlier, SystemC is one approach for translating the floating-point algorithms to fixed-point models and eventually HDL code. Another approach for doing fixed-point modeling is to directly translate the floating-point MATLAB code to HDL and perform simulations using an HDL simulator. As described earlier, this approach creates a chasm between the system engineer and hardware designer because HDL is not a common language between the two. As a result, there is the possibility for errors in implementing the system design in hardware. This direct translation approach also requires a good knowledge of the chosen HDL language and the resources to code and simulate the HDL. Since creating HDL is a part of any prototype design flow that targets either an FPGA or ASIC, the issue becomes how to create it. The fixed-point model and simulation tool chosen in the example rapid prototype design flow is a graphical entry RTL tool called HDS (Hardware Design System), which utilizes the fixed-point library within Cadence's SPW (Signal Processing Worksystem) (www.cadence.com/datasheets/spw.html).

HDS offers the advantage of allowing the graphical entry of an RTL design similar to schematic capture and directly generating the HDL, either VHDL or Verilog, within the tool. This provides a common language for both system and hardware engineers that allows designers without a good knowledge of a particular HDL to design complex systems. For this reason, HDS is a good choice for a rapid prototype design flow because it bridges the gap between system engineering and hardware design. As shown in Figure 8.1, the HDL generated from HDS can also be simulated in any of the various HDL simulation tools on the market, such as Modelsim by Model Technology (www.model.com), which is owned by Mentor Graphics, and NC-VHDL by Cadence (www.cadence.com), prior to RTL synthesis. The one drawback is that the HDL generated within the tool provides few comments and is not easily read, so this VHDL is not readily transferable to other design groups. Since HDS is a component of SPW, mixed mode simulations with both floating-point and fixed-point blocks are also possible enabling the complete verification of system-level designs. The HDL generated within HDS can be directly synthesized to target a specific hardware architecture either an FPGA or ASIC. In the example design flow of Figure 8.1, Synopsys

Design Compiler was chosen as the RTL synthesis tool based on the fact that Design Compiler is commonly used to synthesize ASIC designs from HDL.

The next step in the prototype design flow is to place and route the design on the target hardware using the vendor-specific tools. In the example design flow, Xilinx Virtex-E FPGAs [12] were chosen as the target hardware. Other high-density FPGA vendors include Altera, Lucent, and Atmel [5]. FPGAs are the ideal choice for a rapid prototype design flow. The real advantage FPGAs offer over ASICs is that FPGAs can be reprogrammed as the design changes, which is expected to happen several times in prototyping a new design. Changing an ASIC design requires a complete respin of the ASIC costing time and money. Also, FPGA technology has advanced to the point that very large digital designs can be routed on a single FPGA and run at real-time speeds. In terms of cost, FPGAs are much cheaper than ASICs in low volumes characteristic of prototype designs.

An FPGA essentially consists of several programmable logic blocks, each capable of implementing some arbitrary combinational function. It is these configurable logic blocks, along with the surrounding routing matrix, that are used to implement a digital design. In a rapid prototype design flow, changes in the design need to be quickly verified in hardware so reconfigurable hardware such as SRAM (Static Random Access Memory)-based FPGAs [3] which can be reprogrammed many times make an ideal choice. In terms of speed, FPGAs are well-suited to the implementation of modern communication systems such as the IEEE 802.11a baseband design presented in the case study described in the next section. As specified in the IEEE 802.11a standard, the baseband can support data rates of up to 54 Mb/s. To support this high data rate, the baseband design requires at least 15 GOPS (giga operations per second). One of the fastest DSP processors available in 2001, the Texas Instruments C64x, supports less than 5 GOPS [10]. Current DSP microprocessors, even with advanced architectural extensions like VLIW (Very Long Instruction Word) or super-scalar processing, do not satisfy the arithmetic processing requirements of a modern communication signal processing engine. On the other hand, current FPGAs like the Xilinx Virtex series can support up to 20 GOPS due to the highly parallel processing nature of an FPGA. This makes FPGAs the logical choice for prototyping the IEEE 802.11a baseband design.

Another reason for choosing FPGAs in a rapid prototype design flow is design size. FPGA technology has come a long way in the last five years in terms of the size of the design that can be supported on a single FPGA. The size of digital designs is typically described in terms of the number of "equivalent gates" or just "gates." A "gate" is defined as a two-input NAND gate which equates to 4 CMOS (Complementary Metal Oxide Semiconductor) transistors; so, for example, a 10K gate FPGA design is equivalent to 10K NAND gates or 40K transistors [4]. In 1996, a high-end FPGA could support designs of around 25K gates. This number increased to 100K gates in 1997 and jumped to 2M gates in 2000. The Xilinx (www.xilinx.com) Virtex II family of FPGAs introduced in 2001 can support designs of around 10M system gates [13]. With this system-on-chip capability, current FPGAs can support the more complex DSP algorithms associated with the next generation wireless technologies. Being able to build an SoC design avoids the problems with partitioning the design over multiple FPGAs and/or platforms and makes system integration much easier. For the example rapid prototype design flow shown in Figure 8.1, the Xilinx Virtex-E FPGA [12] was chosen as the target hardware in the rapid prototype case study as it is currently one of the most advanced FPGAs on the market supporting large designs of around two million system gates. There are several over FPGA vendors offering similar SRAM-based FPGAs[5]. For example, Altera (www.altera.com) offers their APEX II family of FPGAs which supports large designs of around 4M gates. The choice of Xilinx over Altera or other FPGA vendors was based mainly on FPGA availability and compatibility with the Aptix system. Using a large FPGA like the Xilinx Virtex FPGA minimizes the partitioning of a baseband design, thereby reducing the complexity and greatly increasing the overall speed of the design. Xilinx place and route software is used to do the final routing of the synthesized netlist on the target Virtex FPGA before porting the design to hardware. The place and route software generates a mapping to the FPGA I/O and binary file for programming the FPGA.

The Aptix MP3C System Explorer (www.aptix.com) was chosen as the FPGA prototype platform in the rapid prototype design flow shown in Figure 8.1. The Aptix system provides the mechanism to program and route the FPGA modules enabling real-time functional verification of a digital design possibly partitioned across several FPGAs. Routing between FPGAs on the Aptix platform is constrained by the bus speeds on Aptix, which are currently around 35 MHz; faster speeds can be achieved by hardwiring the

FPGAs together to bypass the Aptix internal routing. The Aptix system is an ideal choice for a rapid prototype design flow. The Aptix platform supports many of the high-density FPGAs currently available including Xilinx and Altera. The real advantage of using the Aptix system is that it provides for the flexible routing between FPGAs which allows the system and hardware designers to focus on the actual fixed-point design as opposed to the time-consuming board-level PCB (Printed Circuit Board) design and troubleshooting thereby decreasing the prototype development turnaround time. In addition to providing a platform for FPGA-based designs, the Aptix system allows the probing of signals within the design with a logic analyzer. The data captured from the Aptix system on the logic analyzer can be retrieved and compared to the fixed-point design in HDS. Any differences in the hardware implementation versus the fixed-point simulations, shown as comparison results in Figure 8.1, can then be resolved and the design flow can be repeated with a new design. This design flow allows several versions of an algorithm to be evaluated quickly. This last verification step in the rapid prototype design flow essentially closes the loop on the baseband design verification.

In summary, the design flow after the floating-point modeling and simulation in MATLAB is to first model and then simulate the fixed-point design in HDS. After the fixed-point design has been verified, synthesizable VHDL is generated within the HDS tool. This VHDL is then synthesized in Synopsys Design Compiler to target a vendor-specific FPGA. After completing synthesis, the resultant netlist is routed on the target FPGA using the vendor place and route software. The FPGA design files are then downloaded to the Aptix prototype platform using the Aptix software, which supports the download of the design files to the FPGAs, the routing between the FPGA modules, and the programming of the logic analyzer for capturing the data and comparing the data to HDS simulation results. With the aid of scripts, this design flow is very efficient in terms of the cycle-time to get a system-level design to hardware and reach closure in the design verification.

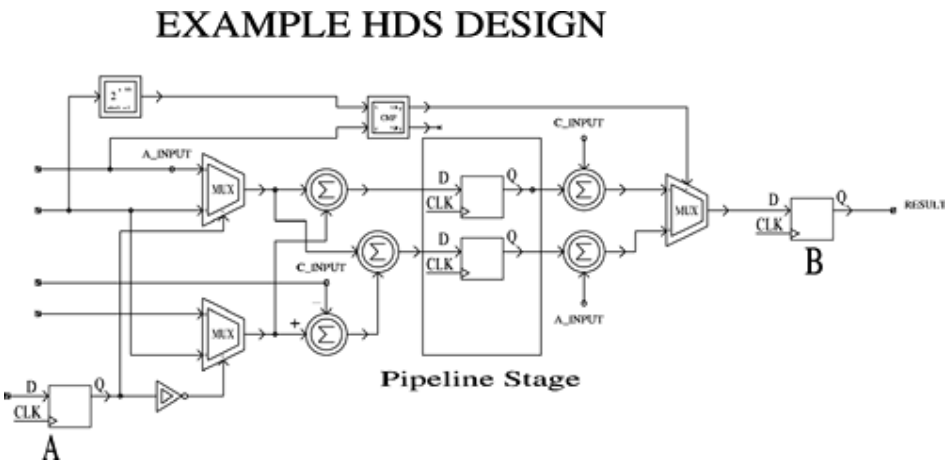
The slowest steps in the rapid prototype design flow described are the fixed-point modeling and simulation in HDS and the tool-driven place and route of the design on the target FPGA(s), which could take from a few minutes to several hours depending on the size of the design. Once a baseline system model is established at the fixed-point modeling level, changes can generally be made to the fixed-point model and verified in FPGAs on the Aptix platform in a single day, which is very fast considering it takes a few months to tape-out an ASIC or design a prototype evaluation board. The rapid prototype design flow is a way to reduce the risk in productizing new technologies. It is a relatively low-cost way to determine the viability of algorithms before committing to production. Projects with tight time-to-market schedules that require real-time verification would benefit greatly from rapid prototyping. A key aspect of the rapid prototype methodology is bridging the gap between system engineers and hardware designers to create a common language that is understood by both groups. This enables a very smooth transition from system-level design to hardware implementation.

Good Digital Design Practices

It is important to understand good design practices to improve the speed and reliability of designs. More extensive coverage of good digital design practices, as well as exercises to help the reader understand the concepts better, can be found in the references for this section. The design practices presented here can be applied to any digital design and should be followed early in the design flow to save time and effort later. These design practices should be considered in doing the fixed-point digital design regardless of whether the target hardware is an FPGA or ASIC. As described earlier, HDS, as part of a rapid prototype design flow, is a tool that is intended to be a common language between system and hardware engineers. Part of this language is understanding some of the implementation issues with regard to speed and reliability of the final design in hardware. Digital design engineers typically understand these implementation issues, but this may not be the case for system engineers. One of the biggest issues in any design is trying to get it to run in real-time. Depending on the design, typical FPGA implementations have maximum clock speeds of 40 to 50 MHz while ASIC implementations have typical operating speeds of 80 to 100 MHz [5]. For this reason, it is important to use good design practices that can help improve design speed especially for FPGAs. Poor design practices early in the design cycle can make a big difference in the final implementation.

One way to improve the speed of a digital design is through the use of pipelining whereby several stages of combinatorial logic are broken into sections separated by registers [3]. Pipelining is a common technique to accelerate the operation of critical datapaths in a design. Pipelining increases the overall speed of a design due to fewer combinatorial stages between registers at the cost of increased latency in the design. An example of pipelining is shown in Figure 8.2 where a pipeline stage is added in an HDS design to increase the speed of the implementation.

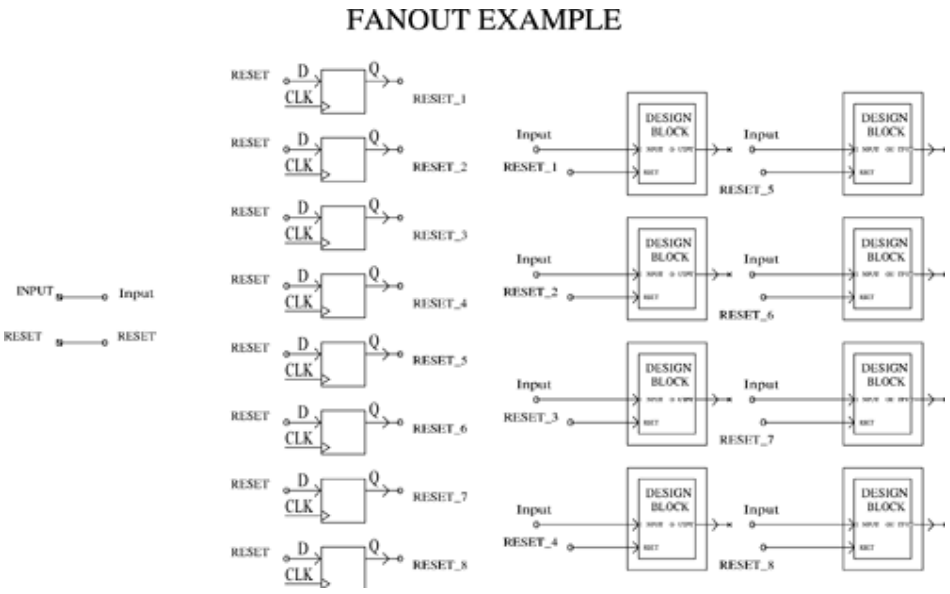
Figure 8.2. An example of pipelining.



This example design without the pipeline stage synthesized to target an FPGA has 826 logic gates at a maximum clock speed after routing of 68 MHz. With the pipeline stage added, the design synthesized to 982 logic gates, and the maximum clock speed increased to 103 MHz. Using pipelining in this example design gave a 51% improvement in speed at the cost of an 18% increase in logic gates. More pipeline stages could be added to this example design to increase the speed even more. Adding extra pipeline stages for high-performance data paths makes sense in so far as the delays of the registers become comparable to the combinational logic delays [3]. Given this small example, it should be clear that using pipelining especially in large designs can give significant improvements in design speeds. This technique was extensively used in the rapid prototype case study described in the next section to achieve real-time system performance.

Another technique that can improve the speed of a design is reducing fanout. Fanout is defined as the number of load gates that are connected to the output of a driving gate [3]. When a gate has a large fanout, the added load can degrade the performance of the driving gate, leading to problems with the speed and reliability of the overall design. One technique to reduce fanout is to insert a buffer between the driving gate and the fanout. A variation of this technique is shown in the HDS design of Figure 8.3 where the 1-bit RESET net is registered eight times and divided into eight separate nets.

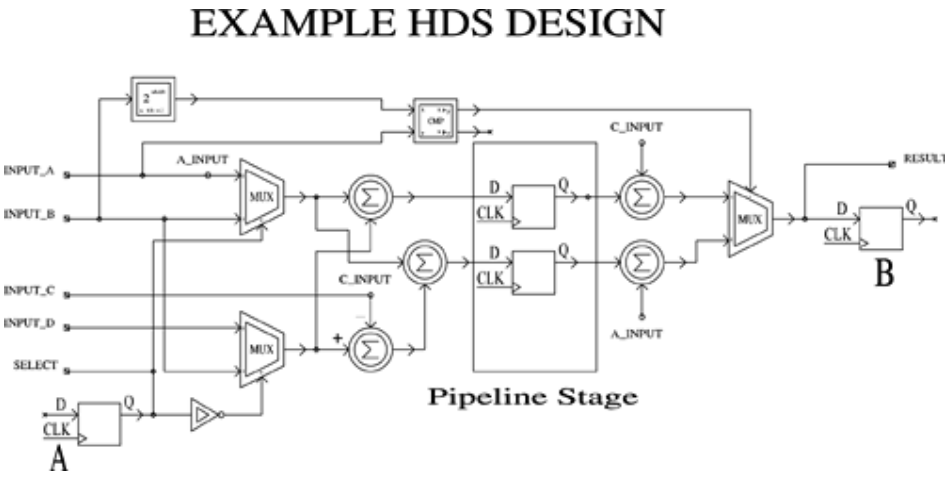
Figure 8.3. An example of fanout.



In this case, the register acts like a buffer to drive the separate nets which reduces the fanout of the RESET net in the design. This technique was used in the rapid prototype case study described in the next section to improve the speed and reliability of the design.

There are some common design practices that can ensure the reliability of the final design. The first practice is registering the output of each block in a design hierarchy. This ensures the follow-on modules have a known timing budget with which to work [2]. Figure 8.4 shows the example HDS design from Figure 8.2 with the output at point "B" tapped directly off the output of the multiplexer as opposed to the register making the timing of the follow-on modules dependant on the variable combinatorial delay of the output as opposed to the known timing of the register. By registering the output, the designers of the follow-on modules have a known timing budget with which to design.

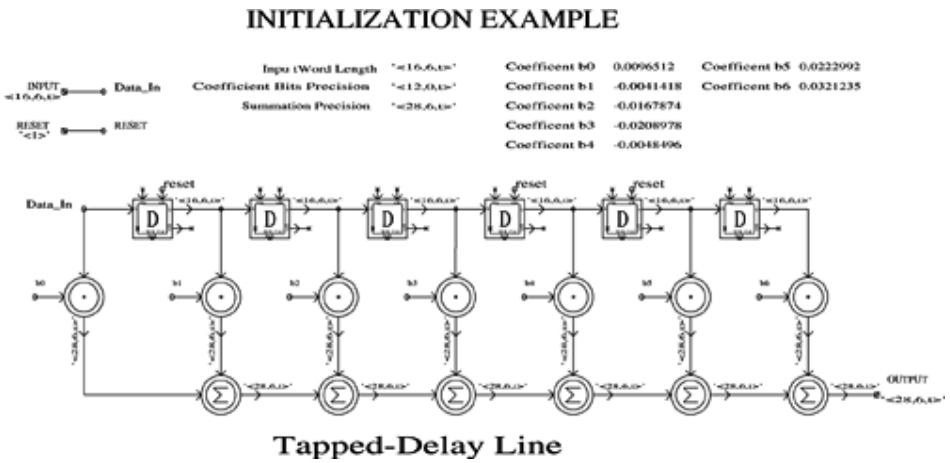
Figure 8.4. An example of unregistered input/output.



By not registering the output of a hierarchical block, logic synthesis becomes more complicated because appropriate timing constraints will need to be specified for all destinations receiving this output signal. Another design practice that should be avoided is having uninitialized storage devices in a design such as registers or RAMs. When storage devices are uninitialized, the initial state is unknown leading to possible

errors in the implementation of the design. Different simulation tools handle the floating initial state problem differently. The HDS simulation tool used in the rapid prototype design flow presented earlier initializes all storage device states to 0, thereby hiding the problem of uninitialized blocks. Most RTL simulation tools, however, use what is called MVL (multi-value level) logic, which includes not only 0 and 1 but also indeterminate (X) and undefined (U) values. This allows designers to more easily troubleshoot this problem. This problem can be very difficult to debug in hardware because the initial state can vary from one implementation to another causing random errors. An example of a FIR (Finite Impulse Response) filter implemented as a TDL (tapped delay line) design with uninitialized registers is shown in Figure 8.5

Figure 8.5. An example of a design with uninitialized registers.



Without resetting all the registers in the design, an unknown state is induced at the output of the filter. Although any initial errors in the filter would disappear after a few clock cycles, the uninitialized filter would no longer be linear or time-invariant.

The design practices discussed are well-known techniques for increasing the speed and reliability of a digital design and should be followed when appropriate to improve design speed and save time in the debugging phase. Using good design practices early in the design process can save a lot of effort later in the implementation phase. All the techniques described were applied to the design in the case study in the next section to achieve real-time performance on FPGAs for the 54 Mb/s IEEE 802.11a baseband implementation.

Rapid Prototyping of a WLAN System

In this section, the rapid prototype methodology presented in the previous section is applied to the design of an IEEE 802.11a baseband radio. Briefly, IEEE 802.11a is a standard for WLAN radios in the 5GHz unlicensed band [1]. The standard based on OFDM (Orthogonal Frequency Division Multiplexing) uses 48 data subcarriers in a 20 MHz bandwidth to support data rates of 6 to 54 Mbps. The topic of OFDM and the IEEE 802.11a standard is covered in detail in earlier chapters. Following the rapid prototype design flow of Figure 8.1, the fixed-point design for the baseband radio is done in HDS based on the floating-point model and simulation in MATLAB. Once the HDS design is verified, VHDL is generated for each design block, which is then synthesized at the RTL synthesis stage to target an FPGA. After the place and route of the resultant netlist on the target FPGA, the design can be downloaded directly to the target FPGA for verification against the HDS model.

After some initial timing analysis in HDS, it was determined that the clock speed for the design needed to be at least 60 MHz to perform real-time signal processing. Reaching the target clock speed of 60 MHz would be a challenge in FPGAs since like most DSP designs, the WLAN baseband design uses several complex arithmetic operations that can slow down the final implementation. The RTL generated for the

fixed-point arithmetic blocks (adders, subtracters, multipliers, and dividers) modeled in HDS are all combinatorial blocks where each operation is performed in one clock cycle. This results in a much slower implementation due to the number of operations that must be performed per clock cycle. A highly pipelined architecture in which the arithmetic blocks have registered outputs as well as internal registers between intermediate operations results in a much faster implementation at the cost of a few clock cycles of latency depending on the operation.

In order to meet the real-time speed requirement of 60 MHz for the IEEE802.11a baseband implementation in FPGAs, it was absolutely necessary to use a highly pipelined architecture in the design. To achieve this goal, most of the arithmetic blocks in HDS were replaced with FPGA core library math blocks, which are pipelined and optimized for the specific FPGA architecture. In HDS, these cores are modeled as "black boxes." Inside these "black boxes," the behavior of the actual cores are modeled for simulation purposes only and are not expanded until the FPGA place and route stage in the design flow. At the place and route stage, the actual netlist for the cores replace the "black boxes." The HDS simulations are bit and cycle-true simulations of the actual hardware implementation, so it is important to accurately emulate the timing and latency of the core "black boxes." If the core behavior is not modeled correctly in HDS, the resultant implementation will have timing errors causing unnecessary iterations through the design flow.

In the IEEE 802.11a prototype implementation, a lot of time was spent going through the design flow to optimize the design for speed and fixing timing errors in the FPGA implementation. In terms of optimizing the baseband design for speed, following good digital design practices, like pipelining, and using the FPGA core library blocks contributed to a 300 to 400 % improvement in overall design speed.

The motivation for choosing a 60 MHz clock speed was based primarily on the processing time required for the 64-point complex FFT (Fast Fourier Transform) block, which is used in both the transmitter and receiver of the IEEE 802.11a baseband. To eliminate the need to design the FFT in HDS, an FPGA core library FFT was chosen. Using an FFT core, provided a netlist which is optimized specifically for the target FPGA and guaranteed to run at 60 MHz. The IEEE 802.11a standard defines the IFFT / FFT period to be 3.2 microseconds, which is very difficult with current DSP technology due to the serial processing nature of DSPs; this is one reason FPGAs were chosen as the target hardware in this design flow. The core FFT provides a result vector every 192 clock cycles which equates to 3.2 microseconds at 60MHz. The chosen 64-point transform engine employs a Cooley-Tukey radix-4 decimation-in-frequency (DIF) FFT [7] to compute the DFT of a complex sequence. The input-data for the core is a vector of 64 complex samples where the real and imaginary components of each sample are represented as 16-bit 2's complement numbers.

In addition to the FFT core, another core that is used extensively in the baseband design is the core RAM and ROM blocks. The Xilinx Virtex FPGA chosen as the target hardware in this design has several on-chip RAM blocks (80 KB for the Virtex 2000E), which are optimized for speed; however, the RTL generated from HDS for RAM and ROM blocks is generic and does not specifically target these on-chip RAM blocks. So, the RAM blocks in HDS must be substituted with Xilinx core RAM blocks using the "black box" approach discussed earlier. To optimize the design for speed, this rule was also applied to all RAM, ROM, and dual-ported RAM blocks in the design.

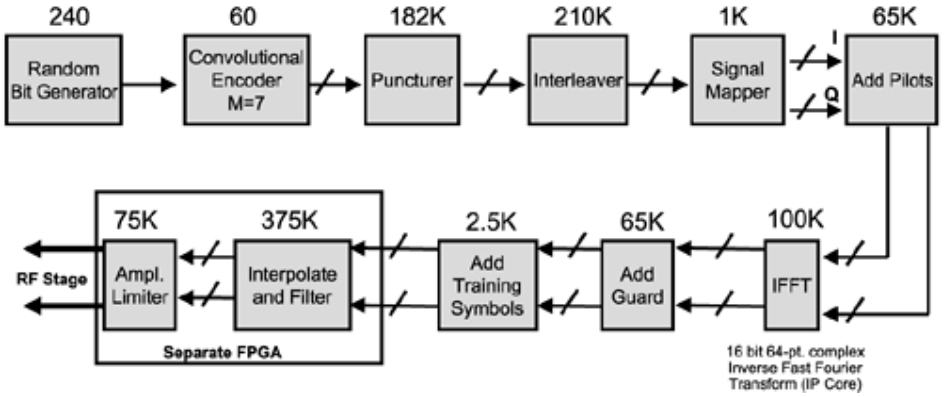
A small group of research engineers was able to successfully implement the IEEE 802.11a baseband in real-time on FPGAs in less than 12 months using the rapid prototype methodology in Figure 8.1. Besides the actual fixed-point design in HDS, most of the effort in implementing the design on FPGAs was focused on optimizing the design to reach the 60 MHz target speed for real-time operation. The specific implementation of the IEEE 802.11a transmitter and receiver is presented in the next few sections.

IEEE 802.11a Transmitter

The baseband transmitter design is a straightforward implementation of the IEEE802.11a WLAN standard [1]. Each subcarrier is modulated using either BPSK, QPSK, 16-QAM, or 64-QAM modulations. The transmitter generates data symbols using these modulations and either a 1/2, 2/3, or 3/4 puncturing rate to support bit rates from 6 to 54 Mbps. The baseband transmitter block diagram is shown in Figure 8.6 with

the estimated FPGA gate count for each block shown above the respective blocks. The transmitter data flow begins with bits being generated from a random bit generator as the input to a convolutional encoder with rate 1/2 and constraint length 7. A random bit generator is used to provide the input stimulus for the system instead of having data from a MAC (Medium Access Control) layer. The output from the encoder is punctured to either 1/2, 2/3, or 3/4 rate according to the IEEE 802.11a mode by removing bits from the encoded data stream. The data is then bit interleaved according to the interleaver pattern specified in the standard. This block is implemented using a LUT (Look-Up Table) ROM block for the interleaver patterns. The symbol mapper block maps the bits to BPSK, QPSK, 16-QAM, or 64-QAM constellation points depending on the mode. The symbol mapper outputs the 48 complex values comprising one OFDM symbol each consisting of an inphase (I) and quadrature (Q) component.

Figure 8.6. IEEE 802.11a baseband transmitter with FPGA gate counts.



The add pilots block adds the four pilot symbols and zero-pads the OFDM symbol according to the standard prior to the 64-point inverse FFT. The inverse FFT outputs a symbol every 192 clock cycles, which meets the standard requirement of 3.2 microseconds with a clock speed of 60 MHz. The inverse FFT output is the time-domain representation of the frequency-domain OFDM symbol. In the add guard block, the last 16 data symbols are cyclically prepended to the beginning of the 64 data samples for a total of 80 data samples in one OFDM symbol. Finally, the preamble block generates the IEEE 802.11a preamble, which consists of long and short training symbols for packet detection, frequency offset, and channel estimation at the receiver. According to the standard, the transmitter is required to generate an OFDM data symbol every four microseconds which equates to a 20 MHz symbol rate. This output rate requires that the baseband transmitter operate at a system clock rate of 60 MHz due to the cycle latency of the 64-point FFT. In a separate FPGA, the output data is interpolated from 20 MHz to 60 MHz to match the sample rate of the DAC (Digital to Analog Converter) in the RF stage. This interpolated data is then filtered using a 24-tap FIR low-pass filter to remove the images generated from the interpolator. Finally, the resultant signal is amplitude-limited to limit the peak to average power of the output signal prior to going to the RF.

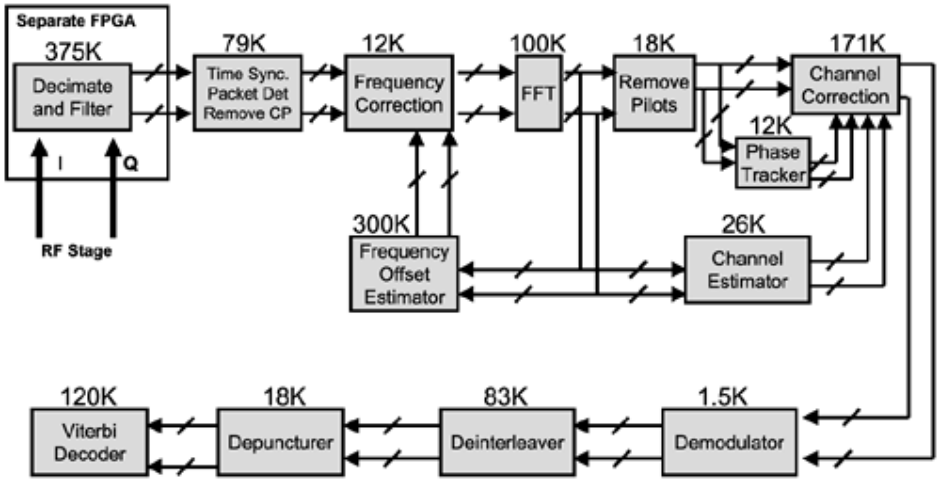
The IEEE 802.11a baseband transmitter shown in Figure 8.6 utilizes a total of 585K FPGA gates which includes 57K logic gates and 528K gates of FPGA core RAM blocks. The design uses 33 of the core 4Kb RAM blocks in a Virtex 2000E FPGA. Maximizing the use of the on-chip RAM blocks decreases reliance on the Virtex FPGA logic gates which are separate from the RAM blocks. The RAM blocks can be configured as single-port RAM, dual-port RAM, or ROM blocks and are used in various parts of the baseband transmitter design.

For example, ROM blocks are used for LUTs in the interleaver, puncturer, and add preamble blocks. Dual-port RAM blocks are used in the FFT core block. The transmitter design of 57K logic gates utilizes 14% of the available logic of a Virtex 1000 FPGA and utilizes 33 RAM blocks, but the Virtex 1000 only has 32 RAM blocks. Based on the number of RAM blocks, the design was migrated over to a Virtex 2000E which has about 33% more logic gates and 160 RAM blocks. In a Virtex 2000E, the transmitter design utilizes 9% of the available logic and 21% of the available RAM blocks. After placement and routing on the Virtex 2000E FPGA, the transmitter meets the 60 MHz system clock speed required for real-time operation.

IEEE 802.11a Receiver

The received IEEE 802.11a packet consists of short and long training symbols, a signal field followed by data symbols. The training symbols are used for packet detection, frequency offset, and channel estimation. The signal field contains the rate and packet length information for the packet. The input to the receiver comes from the ADC (Analog-to-Digital Converter) in the analog baseband. The input from the ADC is 10 bits inphase and quadrature channels at 60 MHz. In a separate FPGA, the data is decimated to 20 MHz and sent through a low-pass filter similar to the one used in the transmitter. The digital baseband receiver block diagram is shown in Figure 8.7 with the approximate FPGA gate counts for each block. The first block in the receiver chain performs the functions of packet detection, time synchronization, and removal of the symbol cyclic prefix or guard interval. This first block detects the beginning of the packet using a windowed moving average correlation with the IEEE 802.11a short training symbols. With a properly set threshold, the algorithm can detect the start of a packet within a few samples. This block generates the processing control signals for the received packet which resets the receiver and triggers the fine-time-synchronizer block also within the first block. The fine-time-synchronizer block determines the exact start point of the received packet's long training symbols by correlating the incoming packet with the known long training symbols. Finally, the cyclic prefix of the data symbols is removed. The block also changes the data rate by buffering one OFDM symbol, so the output is synchronized at 60 MHz which is the processing rate for the baseband receiver.

Figure 8.7. IEEE 802.11a baseband receiver with FPGA gate count.



After the fine-time-synchronizer block, the synchronized signal goes to the FFT block. Like the inverse FFT in the transmitter block, this block performs a 64-point fixed-point complex FFT in 192 clock cycles. The synchronized long training symbols at the output of the FFT are used in the frequency-offset estimation and channel estimation blocks. The frequency offset estimation block estimates the amount of frequency modulation caused by the clock skew between the transmitter and the receiver. It does this by calculating the phase shift in the long training symbols then taking the arctangent of this phase to obtain the corresponding frequency offset. The arctangent operation is performed by using a LUT. A sinusoidal waveform generated with another ROM LUT is then used to generate the compensation signal to the front-end of the receiver to correct the frequency offset in the frequency correction block. The channel estimator block estimates the channel impulse response by comparing the received long training symbols at the output of the FFT with the known long training symbols. The channel estimation result calculated from the preamble is then used for the entire packet. The remove pilots block removes the pilot carriers and reorders the data carriers from the FFT block. The channel correction block corrects the channel distortion by dividing the data carriers by the estimated channel impulse response determined in the channel estimator block.

In parallel, the phase error for the received data symbols is determined in the phase tracker block by comparing the received pilots with the known pilots. The phase error is removed by multiplying the data

carriers by a correction vector. The demodulator demodulates the corrected signal into the four different IEEE 802.11a modes either BPSK, QPSK, 16-QAM, and 64-QAM. The 4-bit soft decisions are determined by the distance between the carrier and several decision thresholds based on the signal constellation. The demodulated soft decision vector then goes to the deinterleaver which uses a LUT to determine the order for the bit deinterleaving reversing the interleaver operation in the transmitter. The depuncturer block inserts dummy zeros in the soft-decision stream according to the designated puncturing scheme either no puncturing, 2/3 puncturing, or 3/4 puncturing. The output of the depuncturer is two 4-bit soft-decision data streams which are the input to the Viterbi decoder. The Viterbi decoder designed in HDS is a 64-state 4-bit soft-decision decoder with a traceback length of 64. The Viterbi decoder uses the input soft decisions to calculate the path metrics; the hard decisions are determined using the traceback through the trellis.

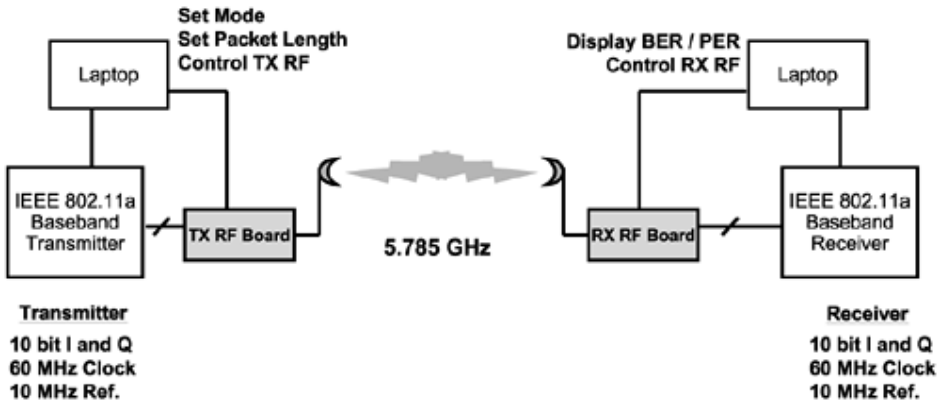
The IEEE 802.11a baseband receiver utilizes a total of 948K FPGA gates, which includes 228K gates of logic and 720K gates associated with the core RAM blocks. The design uses 45 of the core 512 byte RAM blocks in the Virtex FPGA. The FPGA gate count for each block in the baseband receiver is shown in Figure 8.7. Like the transmitter design, the receiver design maximizes the use of the on-chip RAM blocks which decreases reliance on the Virtex FPGA logic gates. The receiver design utilizes 81% of the available logic of a Virtex 1000 FPGA and 45 RAM blocks. The high logic utilization for a Virtex 1000 makes routing difficult, and the Virtex 1000 FPGA has only 32 on-chip RAM blocks. Because of this, the design was migrated over to a Virtex 2000E [12].

In a Virtex 2000E, the receiver design utilizes 52% of the available logic and 28% of the available RAM blocks. A lot of time was spent optimizing the design for speed by reducing fanout and using a more pipelined architecture especially in the HDS implementation of the Viterbi decoder. As a result of this optimization effort, the design speed increased from less than 20 MHz to the target speed of 60 MHz, which enabled real-time data processing.

IEEE 802.11a Baseband Demonstrator

With the completion of the IEEE 802.11a digital baseband design, the next logical step was to integrate a discrete-component RF front end to the design with the goal being to build a complete IEEE 802.11a baseband radio. Using this demonstrator system, it was possible to characterize the performance of the baseband algorithms in an actual system prior to transferring the design to a production group. The IEEE 802.11a baseband demonstrator with the interface to the RF boards is shown in Figure 8.8. It is composed of the IEEE 802.11a digital baseband design implemented in Xilinx Virtex FPGAs on separate FPGA platforms with a parallel port interface to a laptop computer to set the mode and packet length information for the transmitter and to capture the BER (Bit Error Rate) and PER (Packet Error Rate) performance at the receiver. The laptop also has an interface to control the AGC and program the PLL (Phase Lock Loop) on the RF boards to set the LO (Local Oscillator) to 5.785 GHz on the transmit and receive RF boards. The FPGA platform used in the demonstrator is the Aptix MP3C System explorer which provides the programming and routing of the Xilinx Virtex FPGAs used in the design flow.

Figure 8.8. IEEE 802.11a baseband demonstrator.



For the baseband interface to the RF boards, 10-bit Inphase and Quadrature channels with a sampling rate of 60 MHz were chosen for the DAC on the transmitter RF and ADC (Analog-to-Digital Converter) on the receiver RF. Since the output rate of the transmitter is 20 MHz, a 60 MHz sample rate equates to three times oversampling. The LO for the RF boards was specified to be 5.785 GHz which is the upper end of the 5 GHz unlicensed band [1]. Due to the sample rate of the DAC, the data from the transmitter had to be interpolated to 60 MHz which requires the insertion of 0's between data samples followed by the application of a LPF (Low Pass Filter). This LPF was implemented as a 24-tap FIR filter. This interpolation and filter block is followed by an amplitude limiter block which limits the peak-to-average power of the transmitted OFDM symbol. These blocks shown in Figure 8.6 were designed on a separate FPGA as the interface to the DAC on the transmitter RF board. Likewise, in the receiver shown in Figure 8.7, there is a decimation and LPF block on a separate FPGA to decimate the incoming ADC data from 60 MHz down to 20 MHz for the baseband receiver processing. So, there are 2 FPGAs used on the transmit and receive platforms, one for the interface to the RF and one for the baseband processing.

From the digital baseband side, one of the key issues in designing the baseband demonstrator was clearly specifying the digital baseband interface to the RF including ADC output, DAC input, AGC (Automatic Gain Control), sample/reference clock(s), and enable/disable controls. These issues should be addressed during the RF board design and obviously effect the digital baseband design. For the basic IEEE 802.11a baseband demonstrator shown in Figure 8.8, the baseband interface only includes the 10-bit I and Q channels, the ADC and DAC sample clocks, and a 10 MHz reference clock. For simplicity in the baseband demonstrator, all the remaining RF inputs are controlled from a laptop interface, which allows the user to set the AGC level and program the LO. Among the issues to consider in designing the baseband-to-RF interface is DC offset, which could be seen at the output of the ADC due to the cumulative effects of the RF components. This DC offset causes problems in the receiver baseband as it changes the threshold level at which a packet is detected. Since the accumulated data samples from the ADC should add to zero, the DC offset correction can be implemented by accumulating the data samples and subtracting the result from each data sample.

With this IEEE 802.11a demonstrator, it was possible to characterize the performance of the baseband implementation in a working system. Using attenuators after the receiver antenna, it was possible to adjust the SNR (Signal to Noise Ratio) and plot the associated BER and PER. With the demonstrator setup shown in Figure 8.8, where the resultant BER/PER data is captured and processed on a laptop at the receiver, it was possible to run five IEEE 802.11a packets per second with up to 4KB of data per packet which is faster than the MATLAB system simulation. Actual BER and PER performance points were generated from this system and compared to the floating-point model in MATLAB. Some interesting results were found when these comparisons were made. In comparing the PER results for the QPSK and 16QAM modes in the final demonstrator, there was about a 5 dB implementation loss which is to be expected in any fixed-point implementation. In the course of developing the baseband radio demonstrator, numerous fixed-point implementation errors were found and corrected in the FPGA implementation while intermittent random errors did not appear until several thousand packets were run through the baseband demonstrator and PER performance was evaluated. These random errors were usually caused by overflow in fixed-point arithmetic operations. In cases where the PER performance of the demonstrator was far worse than expected, the fixed-point algorithms were reevaluated and fixed accordingly. Final performance data gathered from the demonstrator was used to further improve the baseband algorithms and make recommendations to product groups on the deployment of the system prior to production. The development and evaluation of this IEEE 802.11a baseband demonstrator proves the value of rapid prototyping as the design and deployment recommendations could avoid costly production commitments.

The real-time implementation of the IEEE 802.11a baseband demonstrates the viability of rapid prototyping as a means to test new algorithms and make the transfer of technology from research to product development much faster. The complete IEEE 802.11a baseband design was implemented in FPGAs in less than 12 months by a small group of research engineers, which shows the efficiency of rapid prototyping as a design methodology. This baseband design was taken a step further by integrating it with an RF as part of a baseband radio demonstrator. The performance data captured from this baseband demonstrator was used to make recommendations on the viability of the IEEE 802.11a radio system prior to production, which shows the value of rapid prototyping from a business strategy point of view.