System-Level Design Using FPGAs and DSPs

An Example Showing Software-Defined Radio

DSPs have traditionally tackled signal processing applications. But the complement of FPGA-based processing can add levels of performance only possible with custom ASICs, along with configurability and cost savings that far surpass what ASICs can offer.

by Dick Benson, The MathWorks
and Narinder Lall, Xilinx

Signal processing has traditionally been the realm of DSP processors (for low sample rate applications) and ASICs (for high sample rates). Recently the FPGA has emerged as an alternative option for DSP designers. The FPGA is organized as an array of logic elements and programmable routing resources used to provide the connectivity between the logic elements, I/O pins and other resources such as on-chip memory, digital clock managers, embedded hardware multipliers, embedded microprocessors and multi-gigabit transceivers. A large number of transistors are used to support the programmable routing resources.

As a result, an FPGA is a highly configurable device that can be used to construct customized, and yet reconfigurable, data paths to solve the problem at hand. Additionally, the significant combined computational power of dedicated DSP resources such as 500+ multipliers, accumulators, distributed and block memory, and shift-register logic allows the implementation of highly parallel structures that support computational throughput rates approaching that of today's ASICs.

The FPGA approach to signal processing provides a high-performance, highly flexible, miniature silicon foundry at your disposal, with a turn-around time of hours instead of the months or even years required for many complex ASICs.

Software Defined Radios

A software defined radio is a radio made from reconfigurable/programmable general-purpose components. It is the programming of these components that define its operational characteristics. For instance, bandwidth and modulation (ssb, cw, am, fsk, psk, qpsk, etc.) are completely determined by how the reconfigurable parts are programmed, not by hardware such as filters, mixers, amplifiers or other "traditional" components. Figure 1 illustrates the architectural simplicity of this radio.

Although millions of transistors are required, modern semiconductor manufacturing techniques make the cost per transistor less than that of a one turn wire loop serving as an inductor. The radio uses multiple sampling rates for signal processing. The high-speed work is done in the FPGA while the audio bandwidth work is done in the DSP—a rather perfect division of labor, with each signal processing component doing what makes the most sense.

Prior to 1956 there were two primary methods for generating single sideband (SSB) signals: phasing and filtering. Each had pros and cons. The phasing method required a circuit that generated a 90-degree phase shift across the audio frequency range (300-3000 Hz). Deviations from 90 degrees resulted in less suppression in the unwanted sideband signal. The filtering method, using crystal and mechanical filters, eventually won out. In 1956, Donald Weaver developed a third method that used filters and phase shifts, although the filters were low-pass, not band-pass, and the phase shifts were fixed with respect to frequency. However, the technology was not quite up to the task, and the filtering approach worked well, so the new "Third Method" was not adopted.

The main problems of DC offset and filter gain-phase match, which prevented the idea from being adopted in 1956, simply do not exist in today's DSP implementation. Applying Weaver's technique in DSP hardware is quite easy.

Figure 2 shows a block diagram of a simple SSB generator using Weaver's scheme. To provide a recognizable shape to the spectrum, a weighted sum of sine waves is used as the audio input. The first translation and filtering operation centers the desired sideband around DC and removes the other sideband. Then it is a simple matter to up-convert this complex (I & Q) signal to the desired RF frequency. The output of the final two mixers is either summed or differenced to get the upper or lower sideband. The spectra shown in Figure 2 tell the story.

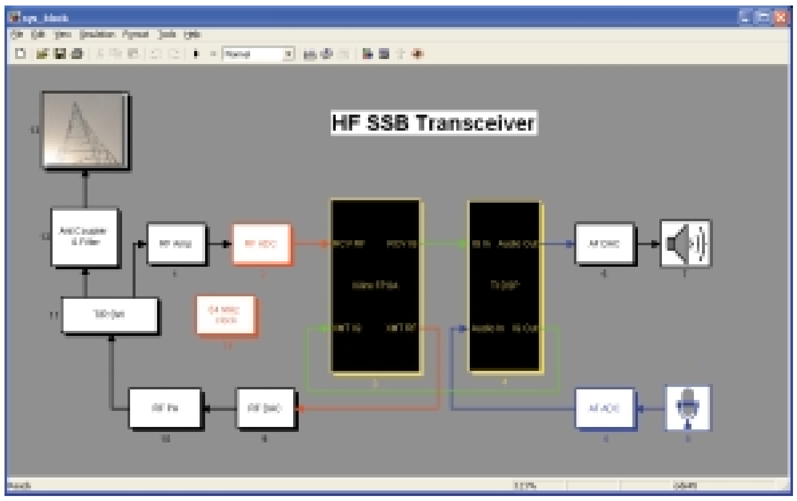

Figure 1 The entire radio architecture can be captured in 12 functional blocks. The black signal paths indicate continuous time signals, while the colored paths (red, green, blue) are discrete time with red indicating the highest sampling rate of 64M Samples/s. The FPGA is used for frequency translation, while the DSP takes care of modulation, demodulation, compression, AGC and other lower data rate functions.

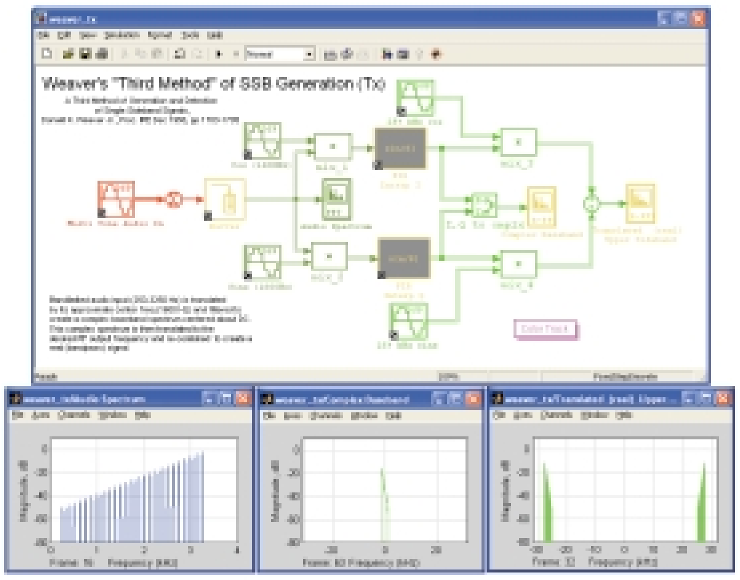

Figure 2 There are several methods that may be used to generate a single sideband signal. The method first proposed in 1956 by Donald Weaver works nicely with today's digital signal processing technology. The above model shows the spectra involved with the evolution of a single sideband signal: audio input (left), translation and removal of one sideband (center), and frequency translation and conversion to a real signal containing the desired sideband (right).

The previous simplified example shifted the upper sideband to approximately 25 kHz. The actual radio will need to go far higher, to almost 30 MHz. To accomplish this, multistage, multirate filtering is required. The MathWorks filter design tools can simplify this design job.

Naturally, to cover the 30 MHz HF spectrum, a sampling clock of greater than 2x30 MHz is needed. The SignalMaster onboard 64 MHz clock satisfies this requirement. For voice communications, the desired audio sampling rate is around 8000 Hz, and a decimation factor of 64e6/8192 = 7812.5 Hz is a good fit. Focusing on the receiver for a moment, the first stage of filtering will be a cascaded integrator-comb (CIC) filter since it maps very well to Virtex-II FPGA hardware. The remaining receiver filters will be FIR decimators or interpolators for the transmit chain. The filter processing is as follows: CIC (D=64), FIR_1 (D=8), FIR_2 (D=8), FIR_3 (D=2) for an overall sample rate reduction of 64*8*8*2=8192. The last filter (D=2) determines the ultimate audio frequency response characteristics. It is trivial to change the filter characteristics since they are all defined by software.

Once the basics have been laid out, more detail can be added. The first step in this process is to create a hardware-independent model. This is can be accomplished using the Simulink DSP Blockset. Figure 3 shows the general signal flow in the radio. The local oscillators are shared by both receive and transmit processing chains. The receiver consists of a digital downconverter followed by a final filter-demodulator stage. The transmitter is simply the receiver blocks turned around with interpolating rather than decimating filters. Once satisfactory simulation results have been obtained, indicating the general signal flow and filtering are correct, it is time to split the model between the FPGA and the DSP. For this design, partitioning the functions is straightforward.

Because the light green blocks in Figure 2 contain data rates that are all at or below 15.625K Samples/s, these sub-systems will be implemented in the DSP chip. The light red blocks have the higher rate data running from 64M Samples/s down to the 15.625K Samples/s rate and will therefore be implemented in the Xilinx FPGA. By the way, this is still the Weaver scheme, but we will describe it in modern terminology as digital down / upconverters with SSB modulators and demodulators.

FPGA Implementation

The Xilinx System Generator for DSP is an extension of Simulink that provides design entry, data path definition, bit- and cycle-true simulations, test bench generation, hardware co-simulation and VHDL code generation. The tool's block library maps to Xilinx DSP LogiCores, which are optimized implementations of typical DSP functions such as filters, direct digital synthesizer and FFT. Engineers pick blocks from the library, define the fixed point parameters (word size, binary point position, rounding, saturation), and hook them together like standard Simulink blocks. Gateways are used to convert between the standard Simulink and the Xilinx-specific data types used by the Xilinx blocks.

As Figure 4 indicates, there are also blocks for Lyr SignalMaster hardware, marked here as "LSP." The ADC (red) represents the 64M Samples/s converter, while the DAC (red) is the 64M Samples/s DAC. The gate_1 gateway block is a 32-bit register that the DSP can write to change the frequency of the direct digital synthesizer in the RF Local Oscillator block. The down-converted IQ stream from the Rx_Mix_Filters is fed to a 32-bit gateway that interfaces to the TI DSP. On the transmit side, another gateway (IQ from DSP) takes data from the TI DSP and drives the IQ input of the TX_Filters_Mix block. This diagram shows the top level; further detail is contained within each block.

Once the data paths and processing are defined, simulation reveals whether the design meets the objectives of dynamic range, and whether spurious responses introduced by the fixed-point implementation have been rejected. If not, chances are that more bits must be used in the filter coefficients and/or data paths. Once the performance objectives are met, you can generate the approximately 200 files of VHDL required to implement the design.

It is then a matter of using the normal Xilinx tool flow of synthesis, place and rout, and bit-stream generation to program the FPGA. This process requires virtually no user intervention and bit-stream generation can be accomplished with a single mouse click.

One of the more interesting reports that can be generated is a "floor plan" of the FPGA design. The digital frequency translator consumed virtually all (95%) of the FPGA. The

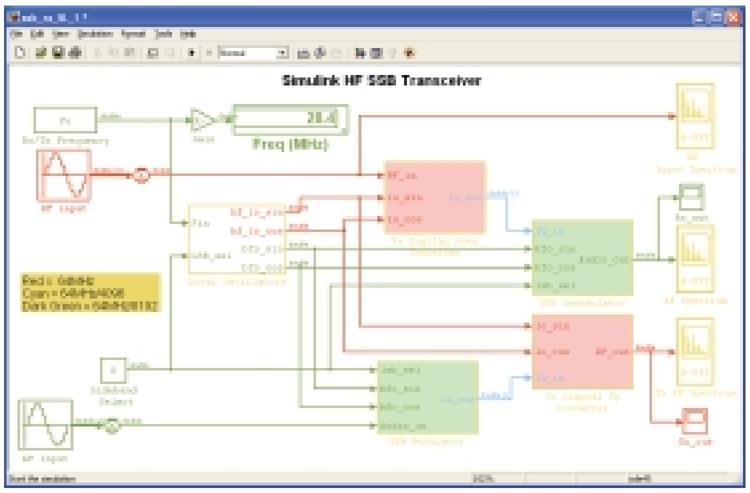

Figure 3 Detail is added to the design to create the transmitter and receiver processing, which shares a common set of local oscillators. This top-level model captures the five main subsystems needed to create the radio. Each block contains several levels of hierarchy, getting down to fixed-point implementation details where required. The light red blocks handle the 64 M Samples/s data rates and will therefore be implemented in the FPGA. The green blocks handle data at both 15.625 and 7.8125K Samples/s, and will be implemented in the DSP chip.

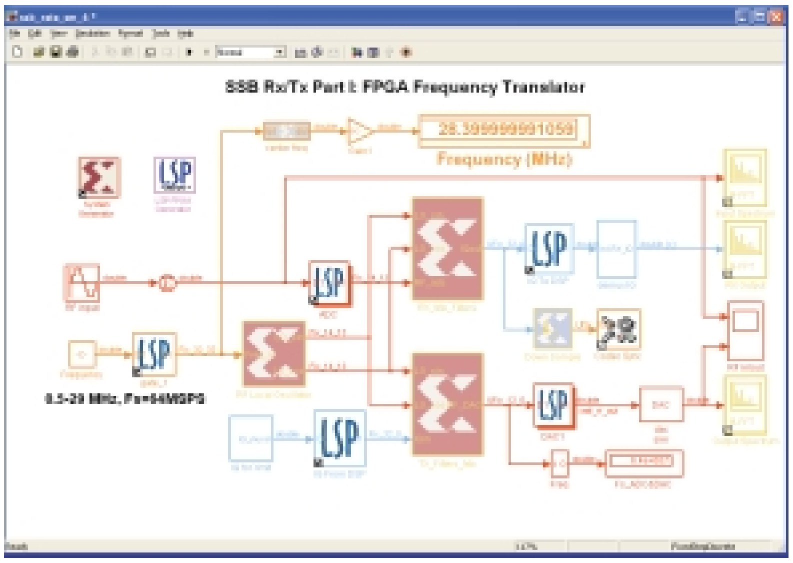

Figure 4 The FPGA signal processing portion of the design is simulated to bit- and cycle-true accuracy by using the Xilinx System Generator for DSP in The MathWorks Simulink environment. Data path size, filter coefficient quantization, and all fixed-point attributes are defined using parameterized blocks. The LSP blocks represent gateways to and from the DSP chip (cyan) while the red block represents the high-speed ADC and DAC hardware on the Lyr SignalMaster development hardware.

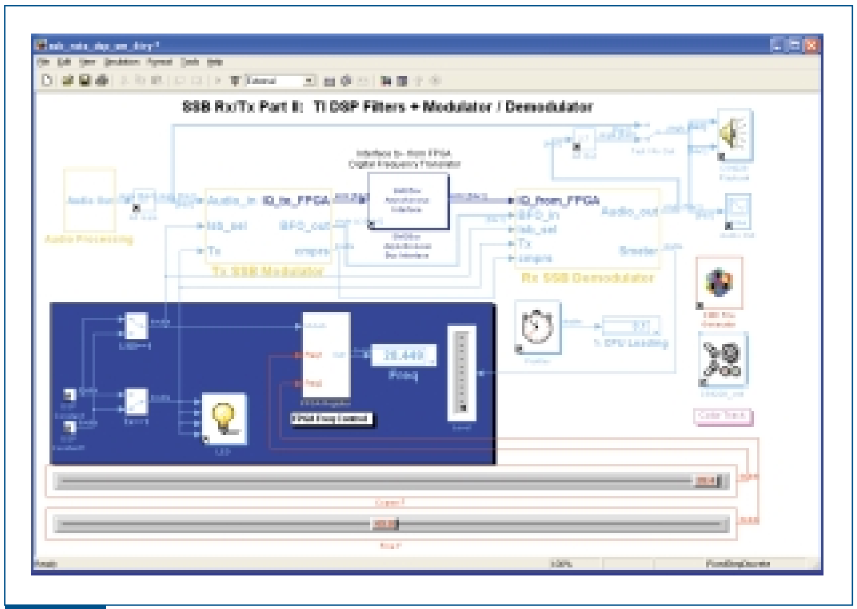

Figure 5 The top-level view of the partition executing in the DSP chip is shown above. The C code to implement the design is automatically generated by The MathWorks Real-Time Workshop option to the Simulink environment. Both the FPGA bitstream and DSP chip binary image are downloaded to the Lyr SignalMaster with a single mouse click. The block diagram then turns into a graphical debugging tool where scopes, meters and numerical displays can show the signals while the target DSP is executing the real-time code. Sliders, switches and other parameters can be changed on-the-fly to control the hardware. This design controls upper/lower sideband selection, transmit/receive, tuning frequency (fine and coarse), audio levels and more.

design did not initially fit in the xc2v1000 part that was on the SignalMaster. Bits were trimmed and both rounding and saturation were abandoned in the transmit data paths. The simulation results indicated that this would not cause any severe problems *if* the proper audio signal conditioning were handled in the DSP chip portion of the design. The ability to easily make these tradeoffs is a key attribute of the design flow and the rationale for using a programmable logic device. In this case, the design fits on the available device, but if we wanted to add more functionality or space efficiency to the system, we could use this analysis to shrink the design further.

DSP Processor Implementation

Figure 5 shows the top level of the radio partition, which is implemented with the TI C6711 DSP chip on the Lyr SignalMaster development hardware. Like the preceding models, this model is hierarchical, and contains details of audio processing algorithms to be implemented on the DSP.

The Audio Processing block contains the low-frequency ADC for the microphone input, a reverberation subsystem, an audio compressor and the ability to generate a two-tone signal for test purposes. The resulting audio data stream is fed to the transmitter (Tx) SSB Modulator subsystem that contains the audio frequency quadrature local oscillator, interpolating filter (I=2), and mixers to generate the transmit IQ data stream. This data stream is then fed to the SMC6xx Interface block as a 32-bit word (16 for I, 16 for Q), which drives the input of the FPGA digital upconverter.

On the receive side, the output of the SMC6xx Interface block is fed to an IQ demux that splits the 32-bit word into two 16-bit words for I and Q. This complex signal then goes though the D=2 decimating stage. The filter output drives an AGC stage that is similar in nature to the audio compression stage. The amplitude-stabilized result of the AGC stage is then fed to the multipliers, which mix the IQ data with the audio quadrature oscillator. The result of this is then fed to an adder. This implements the SSB demodulator.

To control tuning frequency, transmit/receive and sideband selection, a simple user interface was made from switches and sliders. The light bulb icon activates LEDs on the SignalMaster during transmit. A wire from the LED driver is the electrical transmit/receive signal for the external (analog) hardware.

The real-time code to implement the example in Figure 5 can be generated automatically from the Simulink environment using the Real-Time Workshop product. The process that generates the C code can be modified to support a variety of target hardware. A pre-packaged target exists for the Lyr SignalMaster, making it remarkably easy to get up and running. Not a single line of C or VHDL code was manually written.

The block diagram now becomes the user interface to control the radio. The switches and sliders on the block diagram change parameters (T/R, frequency, USB/LSB) on-the-fly while the code is executing in the hardware. Beyond this, parameters can also be changed while the code is running to adjust settings such as reverb level, AGC time constant and signal limiting. Scopes and digital readouts can be used to monitor the signal levels in the system under operating conditions. The system block diagram becomes a graphical tuning, debugging environment as well. ◢

The MathWorks

www.mathworks.com
info@mathworks.com
508.647.7000

91192V00