By Dick Benson, W1QG (v0.1)

An HF SSB Software Defined Radio

Introduction

In the January 1948 issue of QST, SSB was heralded as being "… the most significant development that has ever occurred in amateur radiotelephony... ".

We are now on the brink of another step in technology that will eventually change how HF and all high performance radios will be constructed. This technology will ultimately lead to lower cost, improved performance and vastly increased flexibility.

One of the most remarkable components to evolve over the past decade is the Field Programmable Gate Array or FPGA. The FPGA is uniquely suited for high speed digital signal processing (RF data rates) due to its scalable and parallel hardware resources. As mind bending as the FPGA is, it is made even more remarkable by the latest design tools that allows a high fidelity simulation of the system and then automatically generates the VHDL code to program the device. An engineer's dream, come true!

The same situation holds for the Digital Signal Processing chips. Tools exist that do both system simulation and automatic code generation. And as if all that was not enough, the design environment turns into a graphical user interface to control, monitor, and adjust parameters on your creation while it is running! Yes, it seems too good to be true, but it is real.

This note describes how these tools and components are used to create an HF SSB radio.

Some Definitions

A Software Defined Radio is a radio that is made from general purpose reconfigurable / programmable 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 filters, mixers, amplifiers, and other "traditional" components.

DSP (chip) is a computing device that is optimized for the processing of data streams (signals). The holy grail of signal processing is the "multiply-add" operation since virtually all algorithms need myriad of these computations. In conjunction with a fast hardware multiplier and accumulator, the DSP memory addressing is computed in the same instruction cycle using memory address generation hardware such that two values (coefficient and data) can be fetched from fast memory to feed the multiplier. This allows the multiply-add operation to be done in one machine cycle. Although it is true that a general purpose processor (e.g. Intel Pentium) can in fact computationally outpace the DSP, the power consumption but its cost is commensurate with this performance.

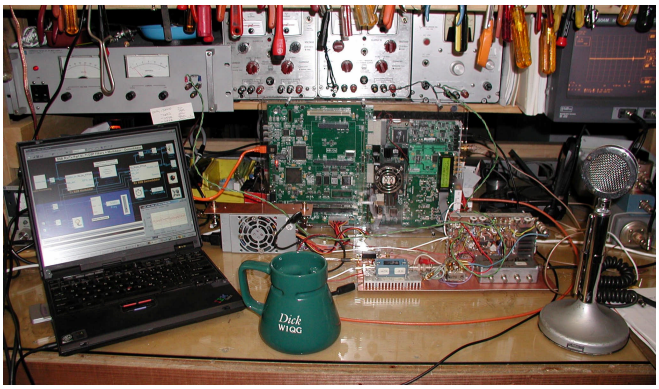

Figure 1— Experimental setup of the Software Defined Radio.

Software on the host notebook computer generates the code required to program the DSP and FPGA hardware the Lyr Signal Processing SignalMaster development board on its edge in the background. After the code is downloaded to the hardware, the notebook also serves as the radio's user interface. The receiver RF pre-amp, transmitter power-amp, anti alias filter, and T/R switching are on the copper clad board behind and to the left of the microphone.

The FPGA is a re-programmable logic device that can contain the equivalent of millions of gates. On board these remarkable devices reside RAM, 18x18 parallel multipliers, registers, look up tables, and the equivalent of general purpose logic. The key advantage the FPGA enjoys over the DSP is its ability to utilize the hardware in a parallel fashion. In other words, one is not constrained to process data streams through one (or at most 4) arithmetic elements, but it literally may have thousands of elements to perform the computations in parallel. This allows the FPGA to process data streams that are running in the 100s of MHz. Until recently, the downside of the FPGA was its difficulty to write the required code to program it. It required knowledge of a Hardware Descriptor Language such as Verilog or VHDL. This has now all changed with the introduction and evolution of the Xilinx System Generator for DSP™.

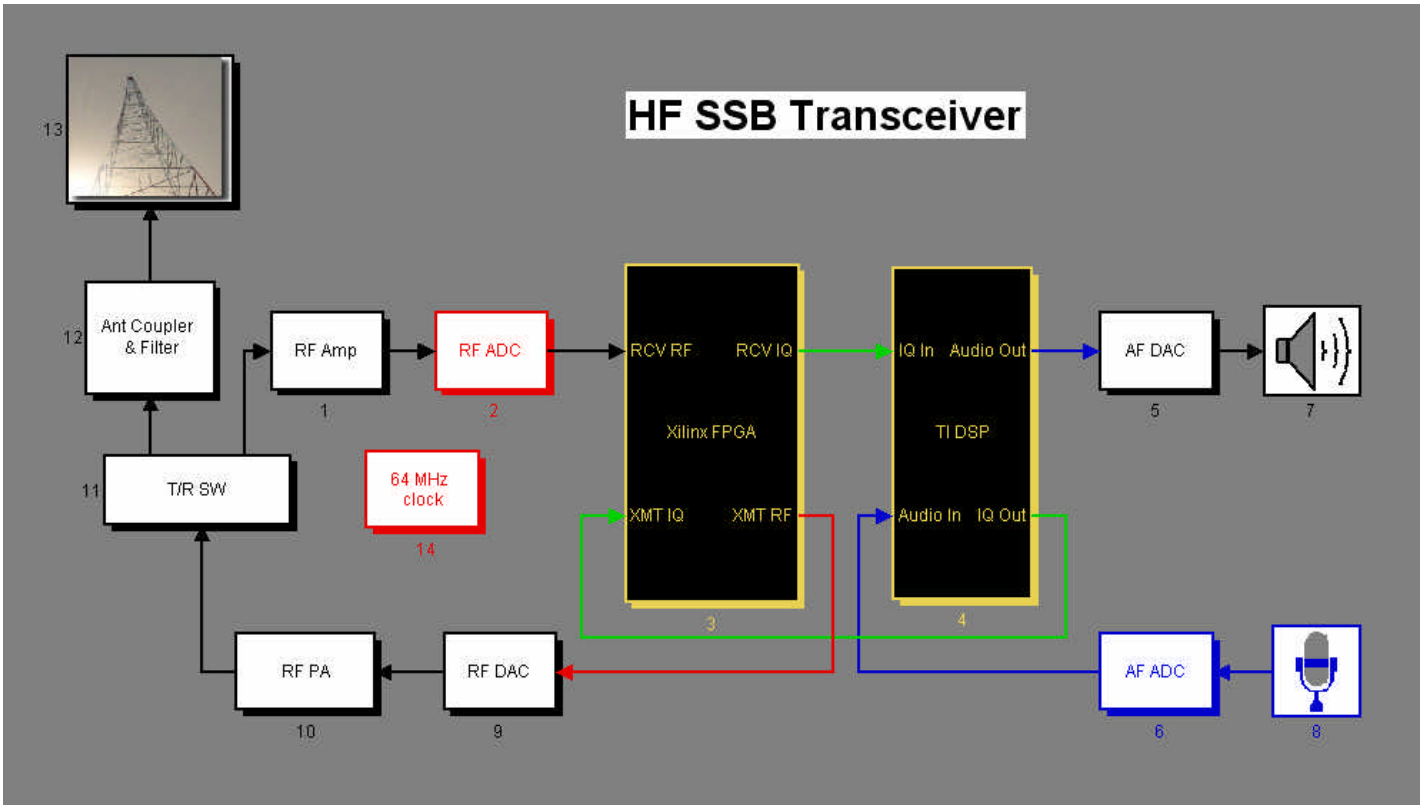

Figure 2— Block Diagram. The radio architecture can be described with only 12 major components. The color code is as follows:

black : continuous time signals
red : the fastest sampling rate, 64 MSPS
green : I/Q data at 64 MSPS/4096 rate
blue : audio data at 64 MSPS/8192 (7.8125 kHz)

Radio Architecture

Figure 2 shows the utter simplicity of this radio! True, millions of transistors are required, but thanks to modern semiconductor manufacturing techniques, the cost per transistor is less than the cost of a 1 turn wire loop serving as an inductor! Who would have thought that to ever be possible? As can be seen, the radio uses multiple sampling rates for the signal processing and 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 component doing what makes the most sense.

Weaver's Third Method

Prior to 1956 there were two primary methods known for generating SSB signals: phasing and filtering. Each of these methods had its pro's and con's. 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 led to a reduction in the unwanted sideband suppression. The filter method eventually won out with both crystal and mechanical filters being used. In 1956, Donald Weaver came up with a third method that apparently had been overlooked. It required filters, and phase shifts, but the phase shifts were fixed with respect to frequency and the filters were low-pass, not band-pass. However, the technology of the time was not really up to the task, and the filtering approach worked well, so the new "Third Method" was not adopted.

Implementing Weaver's technique in DSP hardware is quite easy! The main issues of dc offset, and filter gain-phase match that prevented the idea from being adopted in 1956, simply do not exist.

Figure 3 shows a block diagram, created with the Mathworks Simulink environment, 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 3 tell the story.

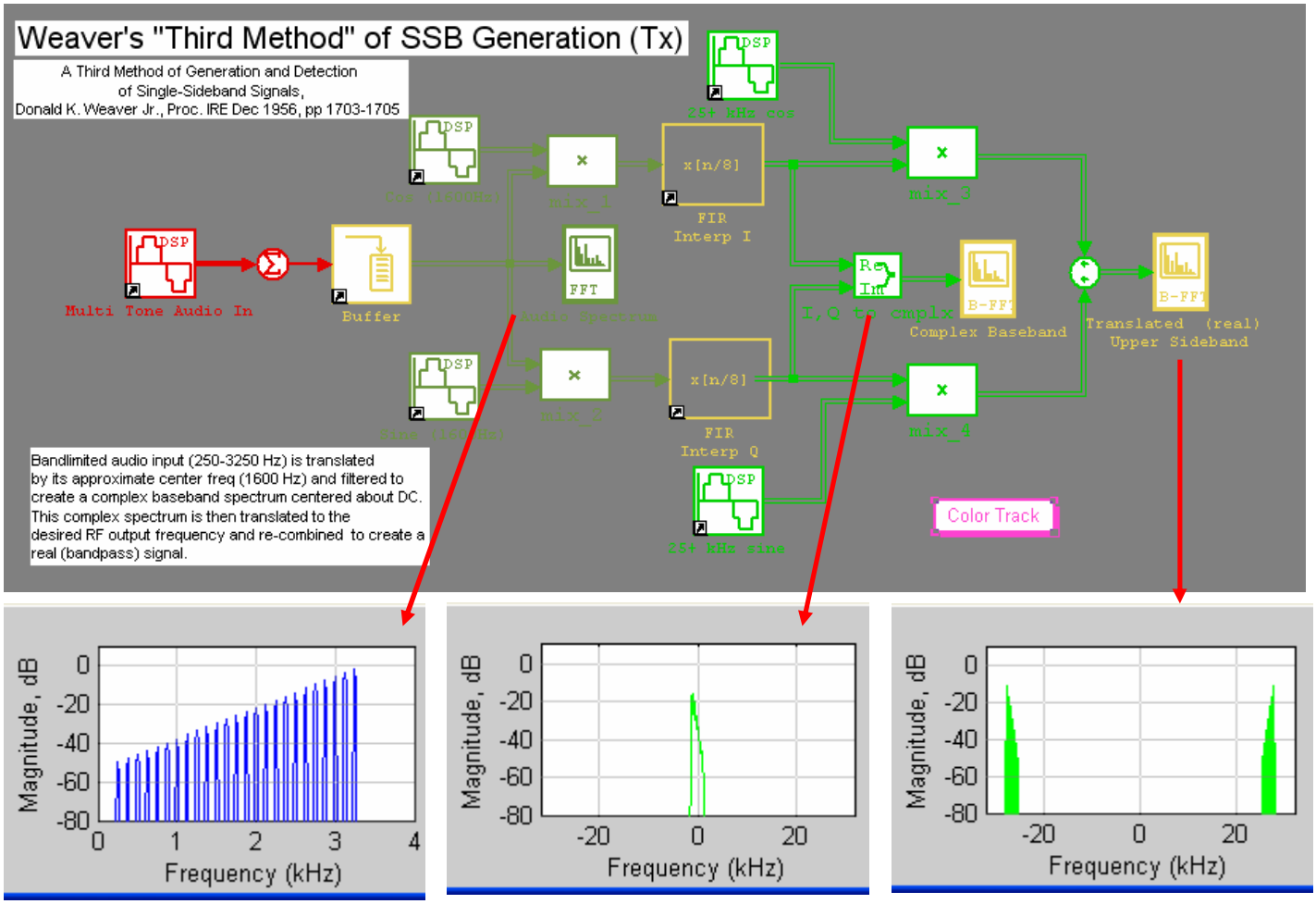

Figure 3— Weaver's Third Method. A simple SSB generator made from Simulink blocks is useful to understand the spectral shifting, filtering, and up translation to RF.

Filter Design

The previous simplified example shifted the upper sideband to approximately 25 kHz. The actual radio will need to go far higher, almost 30 MHz. To accomplish this, multi-stage, multi-rate filtering is required. The Mathworks has a superior set of filter design tools that make this design job quite easy.

To cover the 30 MHz HF spectrum, a sampling clock of greater than 2x30 MHz is needed. The on-board 64 MHz clock satisfies this requirement. The desired audio rate is around 8000 Hz, and 64e6/8192 = 7812.5 Hz is a good fit here. Focusing on the receiver for a moment, the first filter will be a Cascaded Integrator Comb (CIC) filter since it maps very well to FPGA hardware. The remaining filters will be FIR decimators or interpolators for the transmit chain. The filter p-rocessing 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. Figure 4 shows the overall frequency response of the multi stage filter. The same filter coefficients are used for transmit as well. The last filter (D=2) determines the ultimate frequency response characteristics. It therefore borders on being trivial to change the filter characteristics since they are all defined by software.

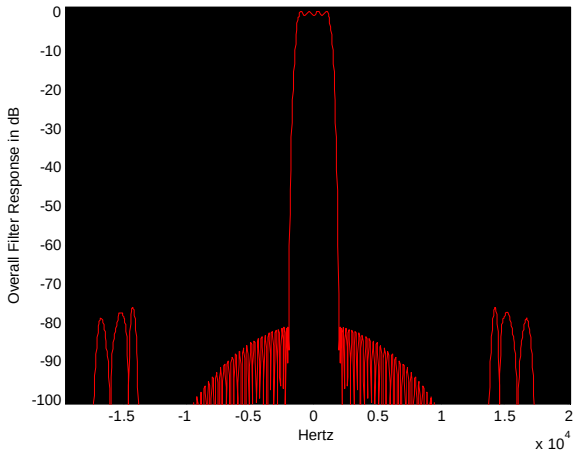

Figure 4— Filter Frequency Response

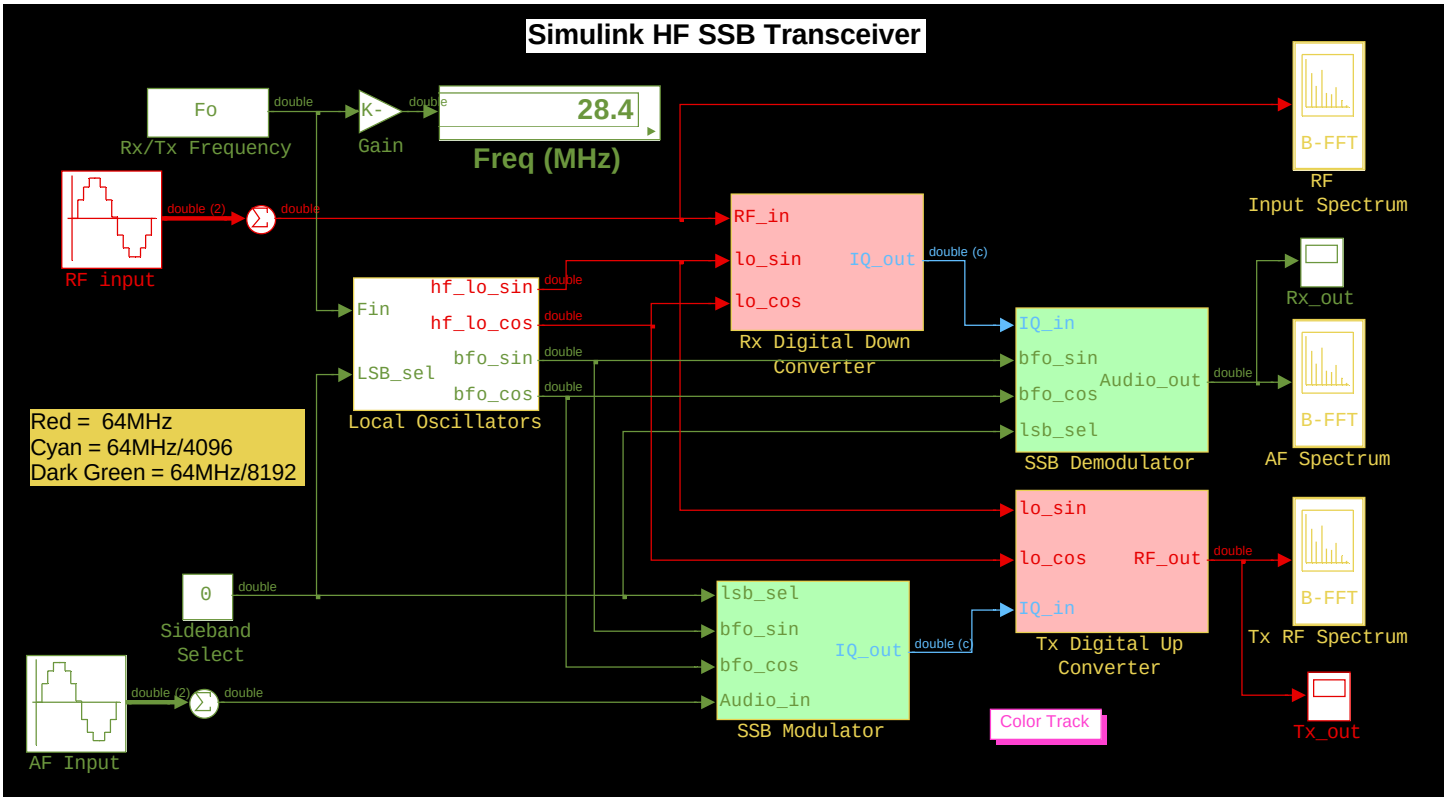

Figure 4— Hardware Independent Model (Top Level)

Evolving the Design

The basics are now laid out, and more detail now needs to be added. The first step in this process is to create a hardware independent model. This is easily done by using Simulink's DSP Block Library. Figure 4 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 down converter 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, it is quite straightforward to partition the functions. The light green blocks will be implemented in the DSP chip while the light red blocks will be implemented in the Xilinx FPGA. And yes, this is still the Weaver scheme, but using modern terminology (digital down / up converters).

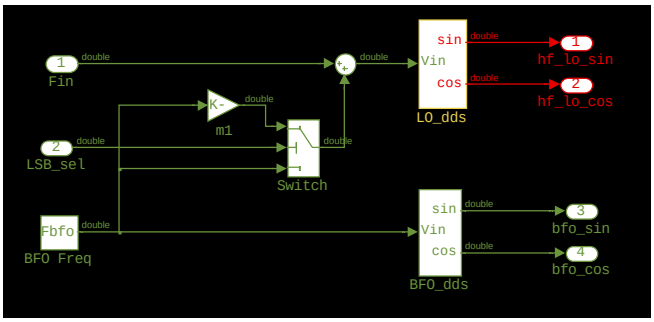

Figure 5— Local Oscillators

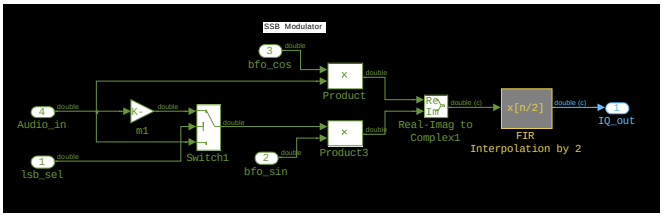

Figure 6— SSB Modulator

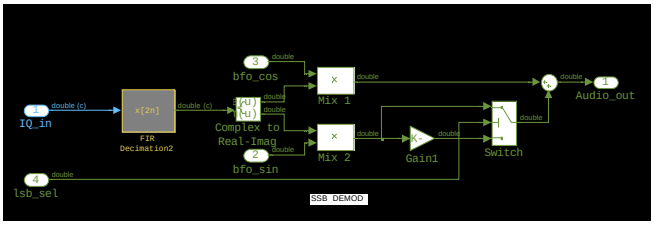

Figure 7— SSB Demodulator

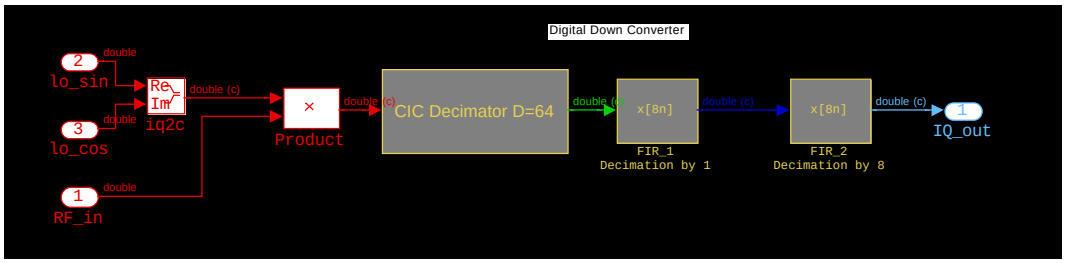

Figure 8— Rx Digital Down Converter

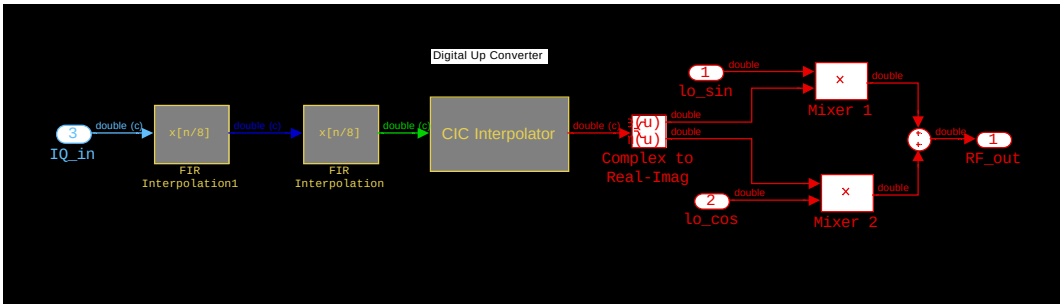

Figure 9— Tx Digital Up Converter

The FPGA Design

The Xilinx System Generator for DSP is a state-of-the-art tool that provides design entry, data path definition, bit / cycle true simulations, test bench generation, hardware co-simulation, VHDL code generation, and more! The key to the tool is that the Xilinx system Generator Bock Library maps to Xilinx LogiCores which are highly optimized implementations of typical DSP functions (filters, direct digital synthesizer, FFT, etc.). One merely picks blocks from the library, defines the parameters (e.g. word size, binary point position), and hooks them together just like standard Simulink blocks. Gateways are used to go between the standard Simulink double precision and the fixed point representation used by the Xilinx blocks. Referring to figure 10, there are blocks that are specific to the Lyr SignalMaster hardware and these are marked with an "LSP". The ADC (red) represents the 64 MSPS converter, while the DAC (red) is the 64 MSPS DAC. The gate_1 gateway 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 downconverted IQ stream from the Rx_Mix_Filters is fed to a 32 bit gateway that interfaces to

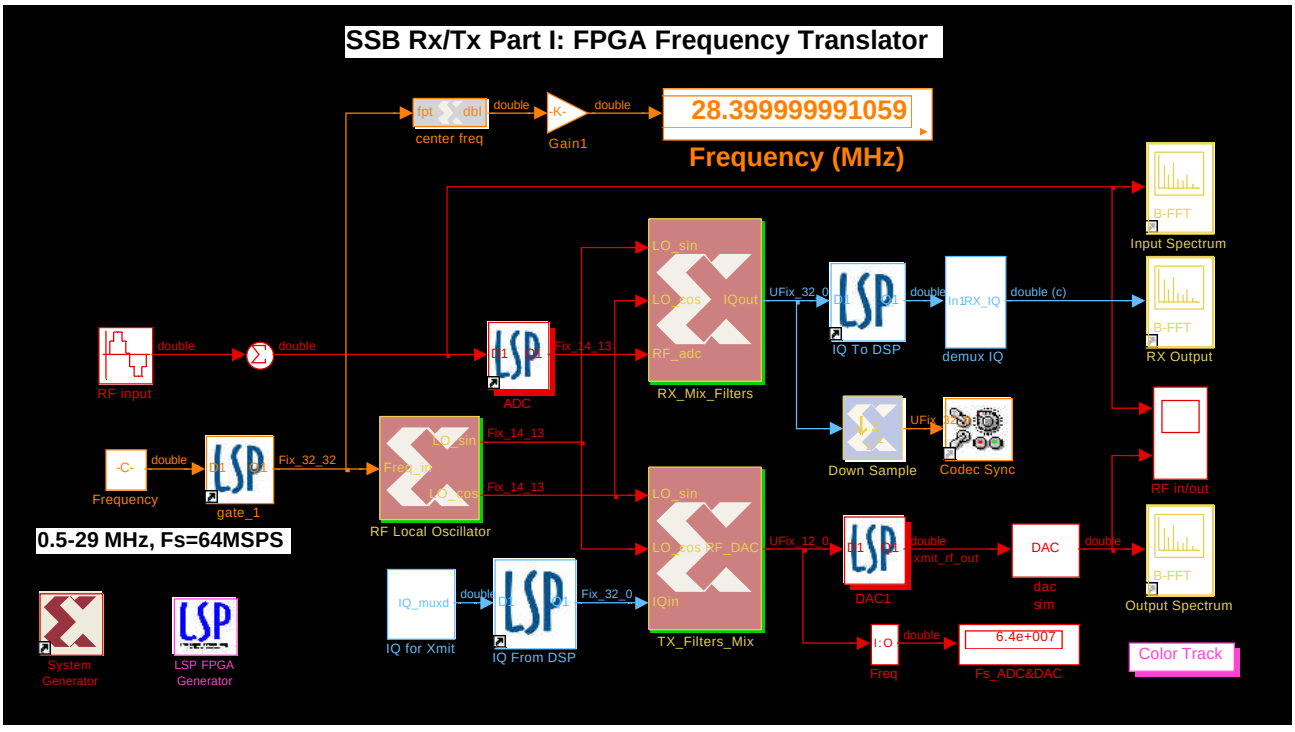

Figure 10— FPGA top Level

the TI DSP. On the transmit side, another gateway (IQ from DSP) takes data form the TI DSP and drives the IQ input of the TX_Filters_Mix block. This diagram is the top level and further detail is contained within each block as shown in Figures 11-13 to the right.

Once the data paths and processing is defined, simulation reveals if the design meets the objectives of dynamic range, and spurious responses introduced by the fixed point implementation. If not, chances are that more bits need to be used in the filter coefficients and/or data paths. Once the objectives are met, a click of the mouse generates the circa 200 files of VHDL required to implement the design.

It is then a matter of using the normal Xilinx tool flow (Figure 14) of synthesis, place and rout, and finally to bit-stream generation to program the FPPGA. This process requires virtually no user intervention and a bit-stream can be accomplished with a single mouse click.

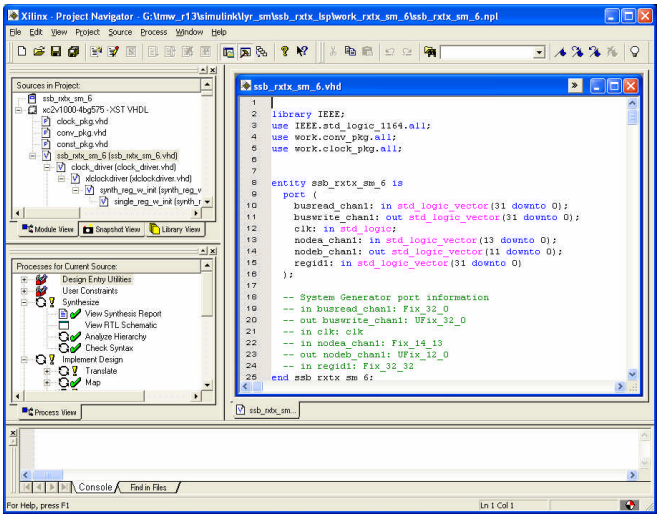

Figure-14 Xilinx ISE 5.2i

One of the more interesting reports that can be generated is a "floor-plan" of the FPGA design. This tool (Figure 15) provides a graphical representation of the FPGA resources used by each major component (DDS, FIR filter etc). As can be seen, virtually all (95%) of the FPGA fabric is used by the digital up/down converter. The design did not initially fit in the xc2v1000 part. Bits were trimmed from the transmit data paths and rounding / saturation were abandoned. The simulation results indicated that this would not cause any severe problems *if* the proper audio signal conditioning were done in the DSP chip portion of the design. The ability to easily make these tradeoffs is a key attribute of the design flow. Much more can be done to shrink the design allowing the use of a lower cost FPGA, but for this example, there is little point.

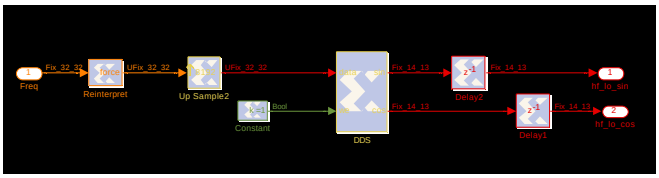

Figure-11 DDS HF Local Oscillator

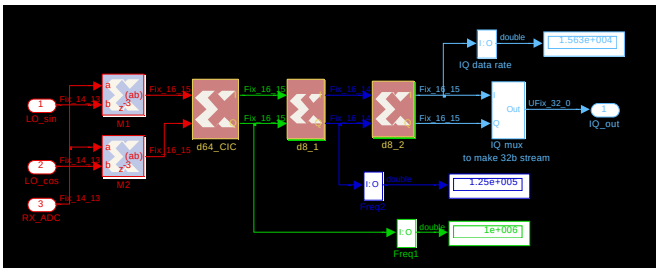

Figure-12 Rx Down-Sampling Filters

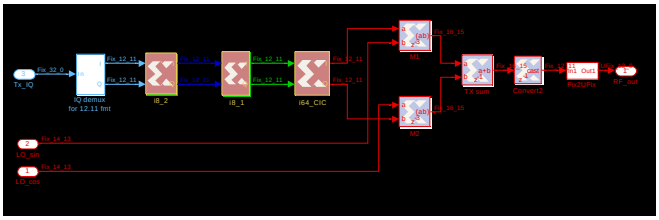

Figure-13 Tx Up-Sampling Filters

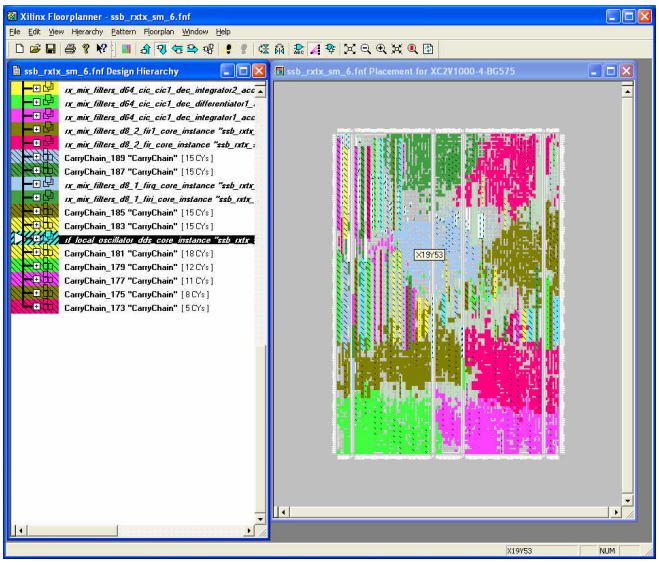

Figure-15 Xilinx Floor Planner

.

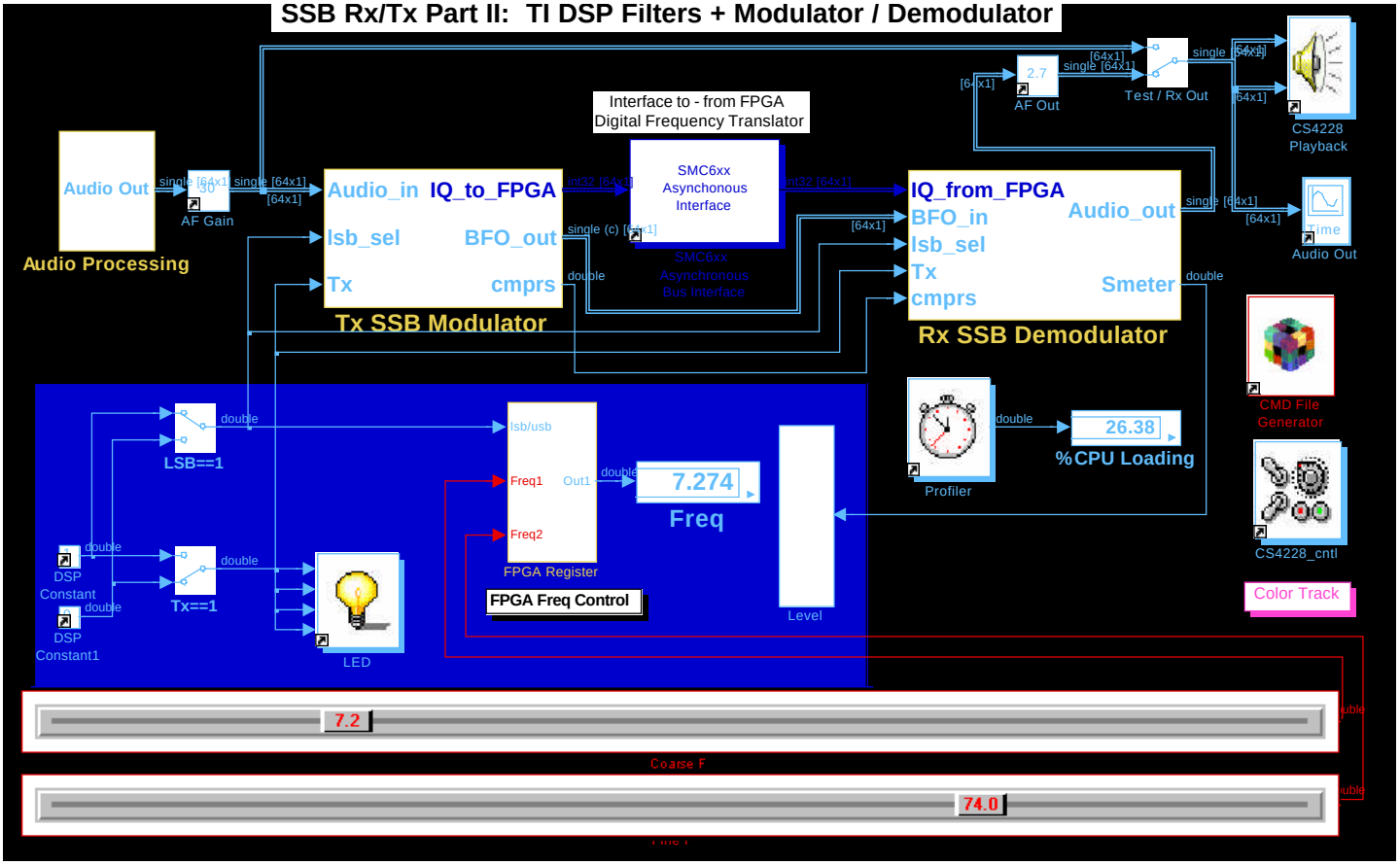

Figure-16 DSP Partition in TI C6711

The DSP Design

Figure 16 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, the model is hierarchical. The Audio Processing block is shown in Figure 17. The left most manual switch selects either the audio ADC in the Mic. Input block, or, two audio sine generators that are summed together to make a built in 2-tone generator. Reverberation is not used with the two tone test, but if the ADC is selected, the output of the Reverberation module is routed to the block output. The details of the Reverberation module are shown in Figure 18.

For processor efficiency, most of the signal processing is done with “frames” of data. On the block diagram, frame based signals show as a wide line. However, feedback systems such as the reverb, need to be implemented on a sample by sample basis, therefore there is an un-buffer and buffer combination to go to sample based and then back to frame based processing.

The transmit chain modulator is shown in Figure 19. The main subsystem is only enabled during transmit to lower the DSP processing load. Figure 20 has the modulator and a compressor which rather than being in the audio chain, is applied to the IQ data stream from the modulator block output.

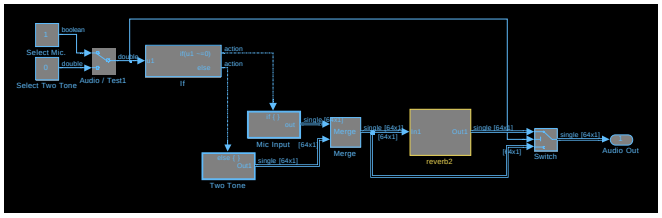

Figure-17 Audio Processing

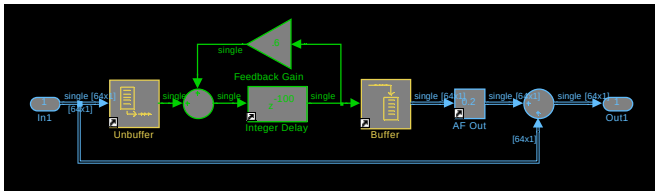

Figure-18 Reverberation Module

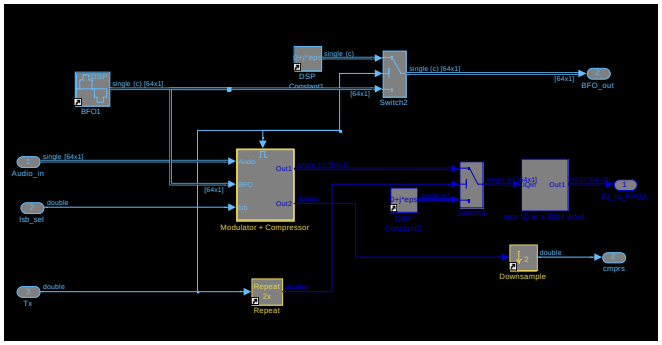

Figure-19 Tx SSB top Level Modulator

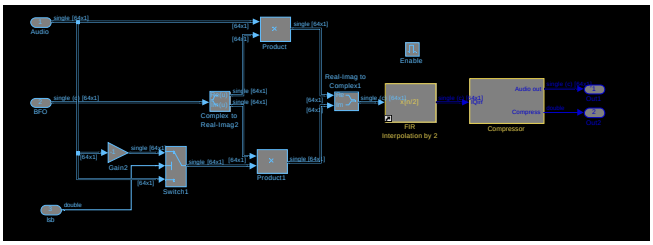

Figure-20 Tx SSB Modulator, Filter and Compressor

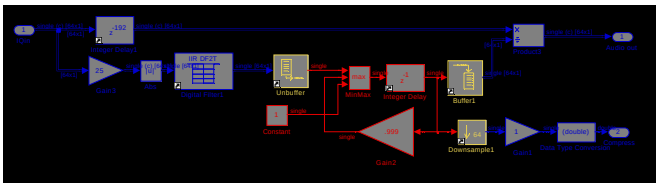

Figure-21 Tx IQ Compressor

Referring to Figure 19, the output of the Compressor feeds a switch that only passes the IQ stream on to the subsequent blocks during transmit. The final block in figure 19 converts the complex single precision IQ data stream into a 32 bit fixed point word (16 bits for I and 16 bits for Q) to be passed to the FPGA by the Asynchronous Interface block in Figure 16.

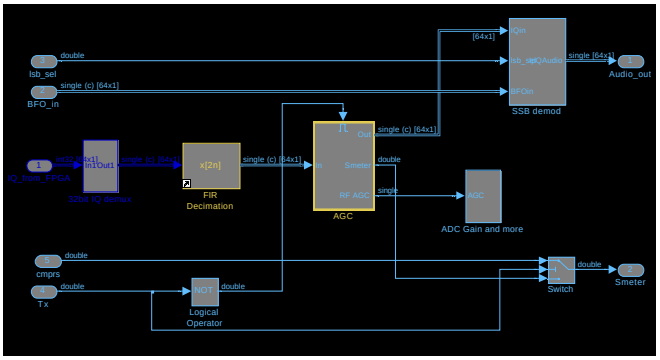

Figure-22 Rx SSB Demodulator

The first block in Figure 22 converts the 32 bit fixed point combined IQ signal from the FPGA to a complex (IQ) single precision float which drives the downsample-by-2 FIR filter. This filter determines the information bandwidth of the receiver. Its up-sample-by-2 counterpart in the transmitter does the same thing. To switch between various information filters, one need only to create them using the DSP filter design tools, and place the filters in “enabled subsystems” in a manner similar to the mechanism used to select between the mic and the two-tone source.

The FIR filter IQ output drives the AGC subsystem which is shown in Figure 23. The AGC is similar to the transmit compressor in that it is a feed forward scheme rather than a feedback scheme. Feed forward is difficult in analog hardware, but easy with DSP. Note that one of the paths (RF AGC) also drives a programmable gain amplifier before the 64 MSPS in the Lyr SignalMasterhardware. Also, some numeric displays are shown measuring signal levels.

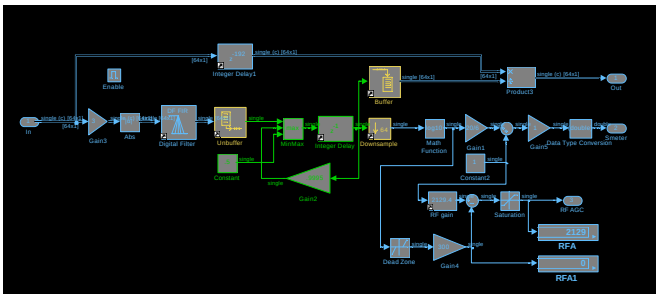

Figure-22 Rx AGC

The normalized IQ level from the AGC subsystem is then fed to the SSB demodulator block along with the BFO signal from the (complex) audio oscillator that was in the Tx SSB Modulator block (Figure 19).

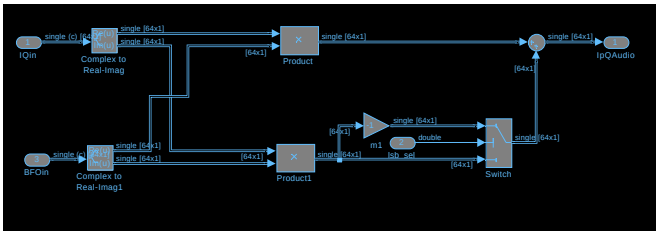

Figure-23 Rx SSB Demodulator

The output of the demodulator (Figure 23) forms the Audio Out signal in Figure 19.

To control tuning frequency, transmit/receive, and sideband selection, a simple user interface was made (Figure 16) from switches and sliders. The light bulb icon turns on some LEDs on the SignalMaster during transmit. A wire run from the LED driver is the electrical Transmit / Receive signal for the external (analog) hardware.

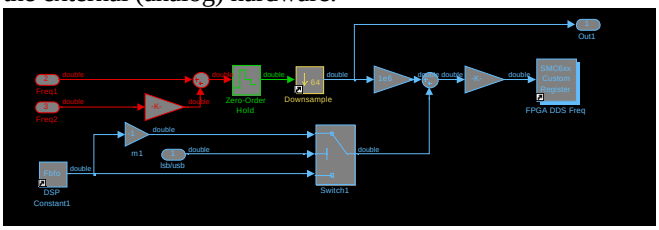

Figure-24 FPGA DDS Frequency Control

The processing of these control signals is shown in Figure 24. The 32 bit word to set the FPGA Direct Digital Synthesizer frequency is computed in this block. Notice that the DDS frequency is offset from the tuned frequency by an amount that is equal to the BFO frequency, and, this frequency shift is up or down depending on the sideband select mode. This places the tuned frequency to the point where there would be a carrier in a non-suppressed carrier system.

Automatic Code Generation and Control

Now the good part! The real-time code to implement the previous block diagrams, download it, and run it, is a mouse click away! A key feature of the Simulink environment is an (optional) fully integrated code generation engine option called the Real Time Workshop. 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 code was manually written.

If that were not impressive enough, 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. Plus, the displays (e.g. numerical in Figure 22) update as well. You can even drop in "scopes" as in the top level of Figure 16, and these update while the code is running as well! Frankly, it has to be seen to be appreciated.

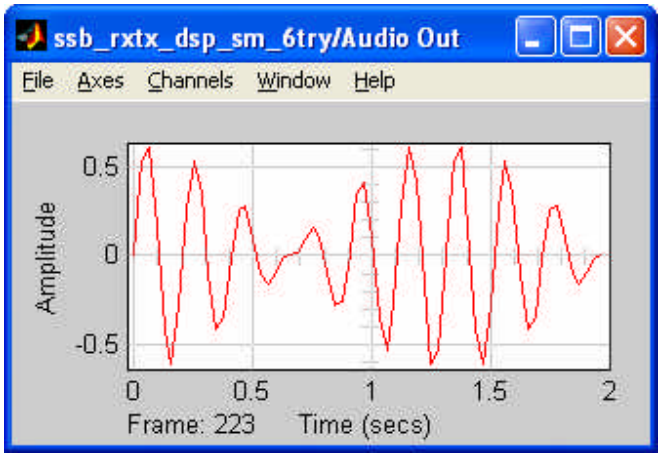

Figure-25 Real time Scope on Audio Signal

Beyond this, parameters (e.g.constants) can also be changed while the code is running to adjust reverb level, AGC time constant, signal limiting etc.

Ok, How Does it Work ?

Reports are universally positive with regard to the signal quality of the radio. In spite of the fact that the receiver has no tuned pre-selector, it overloads only on the strongest local signals. For certain, quantitative measurements need to be made, but there is no question that the radio works well under real world conditions.

Further Work

This design is a Work In Progress. Now that the basic scheme is functioning, additions will be fairly easy. Notice from Figure 16 that only 26% of the processor horsepower is being used for the current code. This implies that 3x more processing is available. On the downside, there has been a pesky intermittent failure which has been found to be related to temperature. New hardware has just arrived and will hopefully cure this ill.

Stay tuned,

Dick, w1qg
September 21, 2003

Useful Links

http://mathworks.com

http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=dsp_software_tools

http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=system_generator

http://www.signal-lsp.com/

http://home.comcast.net/~w1qg/homebrew.pdf