From April 2005 *High Frequency Electronics*
Copyright © 2005 Summit Technical Media

RTDX®-Based Simulation Tools Support Development of Software-Defined Radio

By Robert G. Davenport
MC^2 Technology Group, LLC

This article describes how linked software simulation tools can be used in the design and verification of a software-defined radio that uses popular digital signal processing devices

Recognized simulation software manufacturers, including Mathworks, National Instruments and Eagleware-Elanix Inc., have developed an interface between their tools and Texas Instruments' (TI) digital signal processors (DSP) using TI's JTAG-based RTDX® interface. Since RTDX is common to the TI C5000, C6000 and OMAP DSP families, a single tool can be used to develop software on all of these popular devices. This article presents an example of the use of these tools in the development and verification of DSP algorithms for a Software Defined Radio (SDR).

The SDR discussed in this paper is part of an existing communications radio capable of AM and FM voice and data communication. The algorithms execute on a TI TMS320C5510 DSP. An RTDX interface between SystemView by Elanix® and the target DSP was used to both validate and improve the performance of the existing algorithms. The examples illustrated in this presentation are an FM voice receiver and its IF AGC algorithm.

An FM Software Defined Receiver

The FM voice receiver diagram is shown in Figure 1. The receiver IF input is a 30 kHz IF signal digitized at 120 ksps. The digitized signal first passes through a software AGC algorithm that computes the received signal strength and generates a gain control word. The control word derives an analog voltage that is applied to the radio's variable gain IF amplifier.

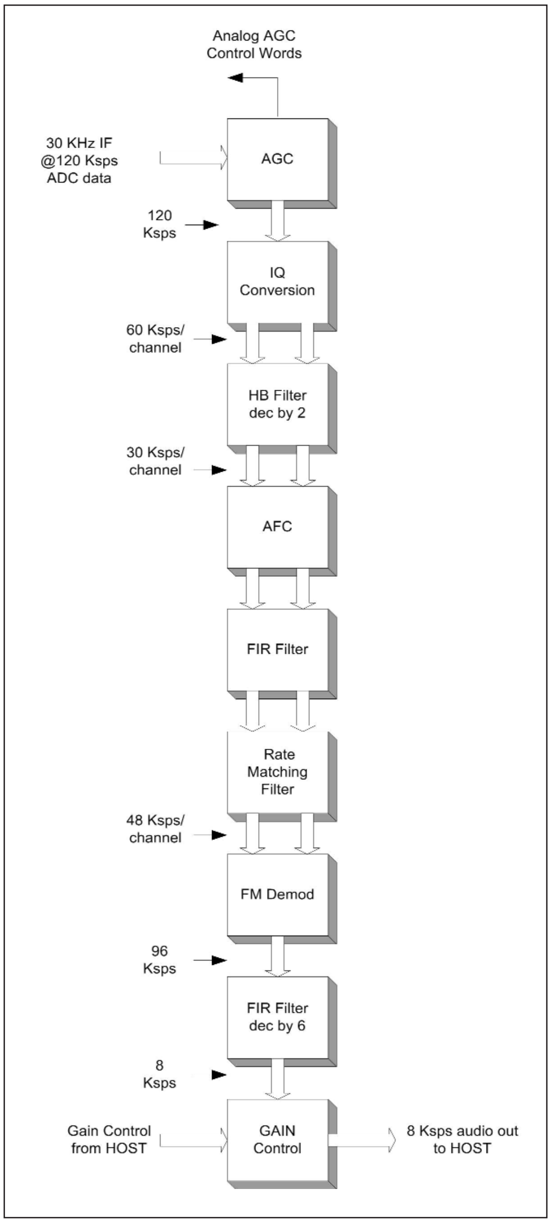
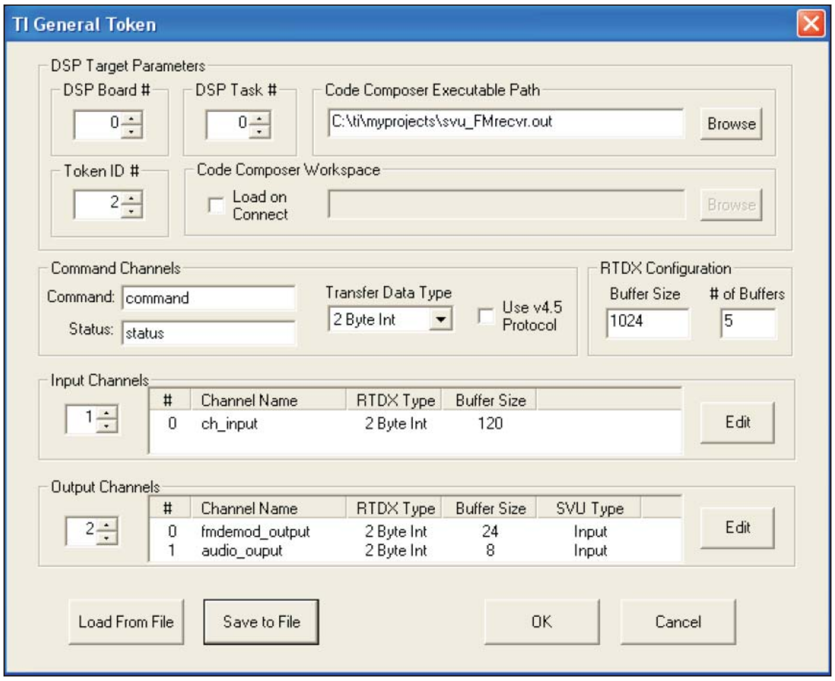


Figure 1 · FM receiver block diagram.



TI General Token

DSP Target Parameters

DSP Board #: 0

DSP Task #: 0

Code Composer Executable Path: C:\ti\myprojects\svu_FMrecvr.out [Browse]

Token ID #: 2

Code Composer Workspace: ☐ Load on Connect [Browse]

Command Channels

Command: command

Status: status

Transfer Data Type: 2 Byte Int

☐ Use v4.5 Protocol

RTDX Configuration

Buffer Size: 1024

# of Buffers: 5

Input Channels: 1

#	Channel Name	RTDX Type	Buffer Size
0	ch_input	2 Byte Int	120

Output Channels: 2

#	Channel Name	RTDX Type	Buffer Size	SVU Type
0	fmdemod_output	2 Byte Int	24	Input
1	audio_ouput	2 Byte Int	8	Input

Figure 2 · RTDA token pop-up window.

Sampling the 30 kHz input signal at 120 ksps results in four samples per cycle of the input waveform. This allows the signal to be separated into in-phase and quadrature (I & Q) components by treating the even number samples as real (I) and the odd numbered samples (with appropriate sign change) as imaginary (Q). Because the samples are separated in this way, the I-Q conversion block yields two data streams each sampled at 60 ksps. This effectively decimates the sample rate by two.

The complex data streams are then filtered by a half-band decimation filter that further reduces the output sample rate to 30 ksps. The 30 ksps streams then pass through a programmable FIR filter that establishes the desired IF bandwidth.

Next, the data streams are passed through an FIR rate matching filter that ultimately reduces the sample rate from 30 ksps to 24 ksps. The 24 ksps I-Q streams are applied to a quadrature FM discriminator, producing a single data stream consisting of the demodulated voice sampled at 24 ksps. The demodulated FM output is further filtered by a programmable FIR decimating baseband filter that decimates the 24 ksps sampled audio to the final 8 ksps sample rate. The 8 ksps sampled audio is passed through a final host-adjustable gain stage before being passed to a host processor for additional audio processing.

SystemView Simulation

The SystemView simulation of the FM receiver is shown in Figure 3. SystemView provides a rich library of user-configurable source, sink and functional blocks. The blocks can be easily interconnected into the desired block diagram by dropping the desired block on the GUI development screen and wiring them together. Double-clicking the block opens up a user screen that allows the user to configure the block as required. Sample rates, block sizes and iteration controls are accessed via GUI pop-up menus on the SystemView tool-bar. The input and output results can be displayed in real-time on the GUI as shown in Figure 3, or each result can be graphically analyzed in detail as shown in Figure 4.

Referring to Figure 3, the simulation begins by generating a 1 kHz sine wave input signal using a Sinusoid Source block. The output of the sine wave source is connected to a graphical display sink that displays the output on the GUI in real-time. The sine wave also drives an FM modulator block that is configured for a center frequency of 30 kHz and a peak frequency deviation of 4 kHz. The system sample clock is set to 120 kHz, so that the FM output signal consists of the four samples per cycle required to perform I-Q conversion in the DSP.

The output of the FM modulator block is summed with a Gaussian noise source block that is used to simulate the receiving system noise figure. Note that the blocks described thus far execute in double-precision floating point, allowing a high degree of accuracy for analog simulation. (The SystemView Communication Library provides several additional channel model blocks that can be used to simulate a variety of RF environmental conditions.)

The FM signal with additive noise is converted to a 16-bit binary word by the DSP Converter block, which simulates the digitization of the received signal. In this simulation, the outputs of the FM modulator and the noise token are set to equal the desired magnitude of the binary input word.

The digital signal is then input to a token that provides an RTDX interface between SystemView and the TMS320C5510 processor. The interface requires the development of a C wrapper interface, examples of which are provided with SystemView. Once the wrapper is developed, it is connected to the SystemView simulation by double clicking on the RTDA (Real-Time Data Architect) token. This brings up the pop-up window shown in Figure 2. The pop-up window is used to identify the path to the TMS320C5510 executable output file to be accessed by SystemView. The pop-up is also used to set the number of input and output channels as well as the buffer sizes and data precision. In this particular simulation, there is one RTDX input channel and two RTDX output channels. The input channel is set to transfer a block size of 120 samples, corresponding to one millisecond of input data. The output channels selected for display are the output of the FM demodulator and the audio output received after the final stage of filtering and decimation.

Referring again to the receiver block diagram in Figure 1, the output of the demodulator is sampled at a 24 ksps rate, while the output of the decimating audio filter is 8 ksps. Therefore the two RTDX output channel buffers are set to 24 and 8 words respectively, since those are the number of output samples that will be produced when 120 input samples are processed by the receiver algorithm. The total number of samples to be processed during a run of the simulation is set by the SystemView clock control. The simulation can be set to run for a specific number of samples, or a single block of samples can be run in successive loops. In this particular system, the sample size was set to 120 samples, and the program was allowed to run for 16 successive loops. (This is a particularly useful technique, because SystemView allows various token parameters to be varied during each successive loop. For example, it may be of interest to increment the deviation of the Gaussian noise token for each iteration of the simulation.)

Once the RTDA token has been properly configured, the simulation can be run. When SystemView starts, the TI development environment, Code Composer Studio, is started and the executable program loaded. The simulation then executes on the DSP until the total number of samples have been processed. During this time, the input and output values are displayed on the GUI as shown in Figure 3. Note the burst of noise on the FM Demod Output shown in Figure 3. When the existing program was initially simulated, this was found to be the result of uninitialized memory. The error was corrected but reproduced for this article in order to demonstrate the program debugging capability of graphically representing the results of the algorithm. In order to debug a problem such as this, it is only necessary to preload the DSP project and output file in Code Composer Studio before running the simulation. It is then possible to set break-points in the program source code and interactively debug the program (step through, display breakpoints and memory, etc.) while running the simulation. Note also that the begin-

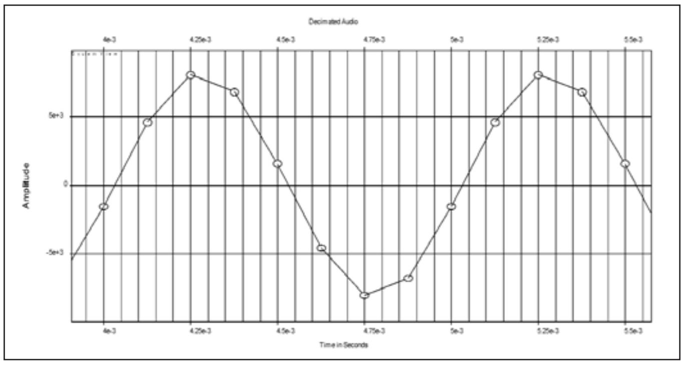


Figure 4 · Expanded display of 8 ksps decimated audio.

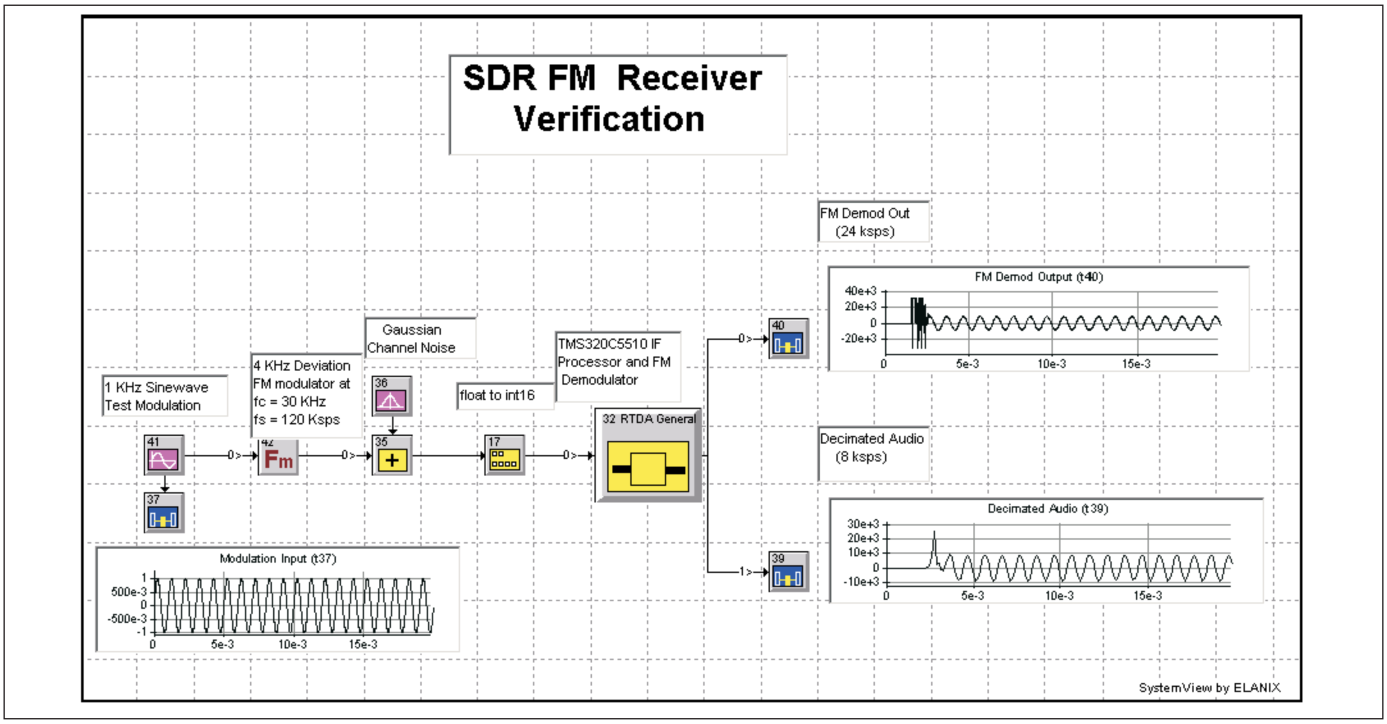


Figure 3 · SystemView FM radio block diagram.

nings of the output waveforms are delayed relative to the input signal. This is because the first 120 samples must be loaded into the algorithm and processed before valid output results are available.

When the simulation is complete, it is also possible to graphically analyze the output results in greater detail using the graphical analysis tools available with SystemView. For example, the 1 kHz demodulated output has been expanded in Figure 4 and the sample points highlighted with circles to indicate the existence of exactly eight samples per cycle of the 1 kHz demodulated output sine wave.

IF AGC Algorithm

The second algorithm described in this article is a simulation of the IF AGC algorithm that is used with the FM receiver. The SystemView block diagram of the AGC is shown in Figure 5.

This particular AGC algorithm is used to control the gain of an external analog IF amplifier, which requires a logarithmic RSSI (Received Signal Strength Indication) voltage as an input. Therefore, the algorithm computes the average power of the received signal in dB and provides the output result as a control word that can be converted to a logarithmic voltage by a digital to analog converter.

This is simulated in SystemView by passing an input signal (in this case a 30 kHz sine wave sampled at 120 ksps) through a SystemView multiplier block (Analog Gain Element) that multiplies the input by the AGC control voltage in order to compensate for increases in gain. The gain compensated input signal is in double-precision floating point, so it is converted to a 16-bit integer format before being transferred to the DSP algorithm via the RTDX channel. Unlike the FM receiver algorithm, which processed 120 samples in each input data block, the AGC algorithm updates by processing every eight samples corresponding to an update rate of 120/8 = 15 AGC updates per 120 sample blocks. Therefore the RTDA token's input block size is set to eight, while the output block size is set to one, since only one RSSI output value is obtained for each block of eight input samples.

The logarithmic output value is represented as a fixed-point 16-bit word which is converted back to a double precision floating point word by the "int16 to float" conversion block shown in the block diagram. In order to simulate the action of the analog variable gain amplifier, the converted logarithmic output value is scaled by an amplifier token (block 14 in Figure 5), offset by a DC gain target (block 42) and inverted by a unity gain inverting buffer amplifier (block 38). The antilog of the scaled inverted signal is computed by raising 10 to the input exponent *x*, by means of block 39 in Figure 5. The converted voltage produced by block 39 is applied to the input of the Analog Gain Element multiplier block, thus closing the AGC feedback loop.

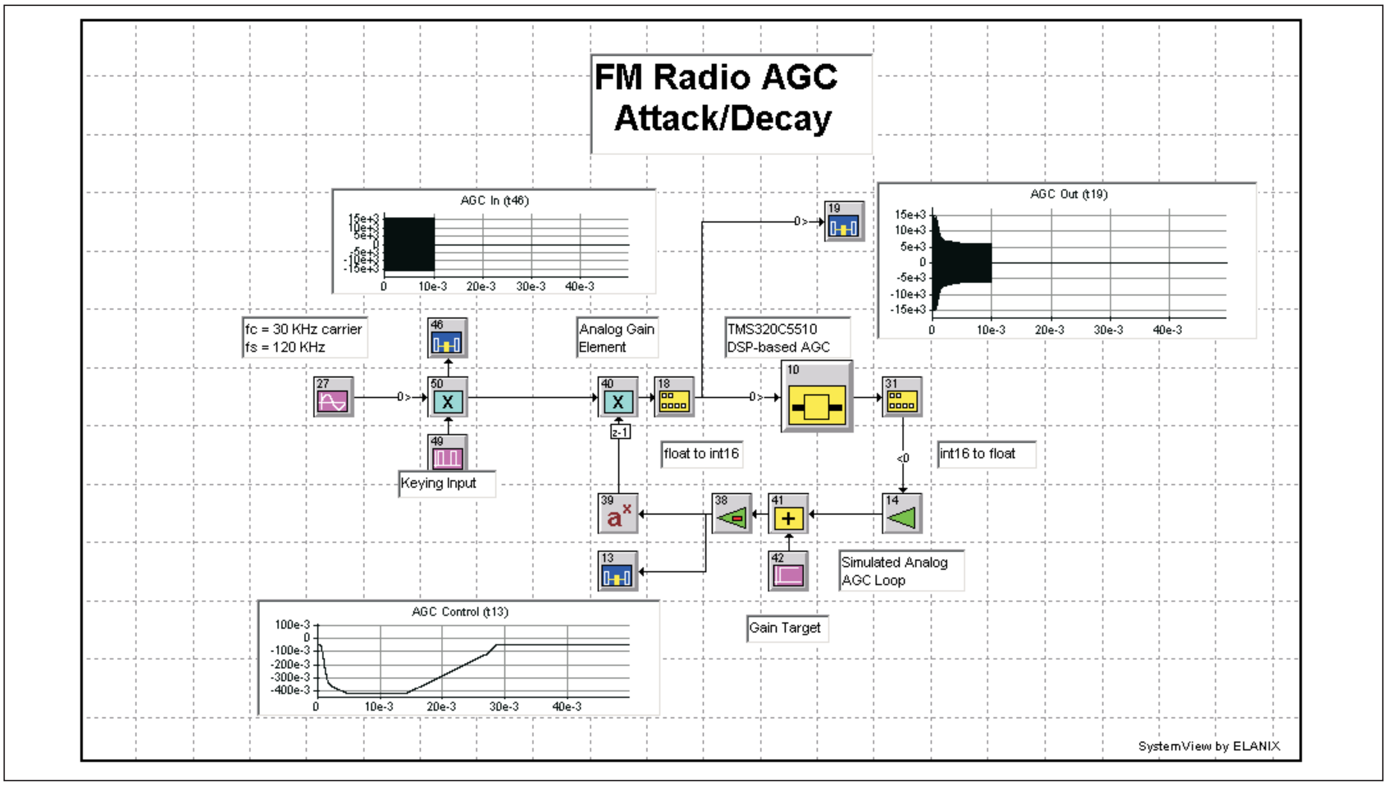


Figure 5 · IF AGC simulation block diagram.

The intent of this particular simulation however, was to measure and “tweak” the attack and decay times of the AGC. In order to accomplish this, the 30 kHz sine wave was input at a relatively high level and “keyed” on for 10 msec once during the simulation in order to observe the attack and decay times. The keying is accomplished by setting a pulse generator block (block 49) to a pulse width of 10 msec, with a repetition interval greater than or equal to the period of the simulation. The output of block 49 multiplies the output of the sine wave generator block which keys the signal during the 10 msec pulse width since the output voltage of the pulse is set to one volt when high and 0 volts when low.

The effects of the algorithm’s attack and decay time settings are clearly seen in the AGC Control output voltage as well as the AGC Out signal display, which plots the input to the AGC Algorithm following the gain element block.

Conclusion

We have presented a method for both developing and validating Software Defined Radio algorithms that are implemented on the Texas Instruments family of Digital Signal Processors. This has been a relatively simple example to illustrate the technique. Ultimately, both the SDR transmitter and receiver algorithms can be simulated this way. Both algorithms are connected together with channel models blocks that allow simulation of the RF environment. This permits analysis of the system performance, especially the effect of error correction algorithms, such as convolutional and Reed Solomon encoding.

Since TI’s Code Composer Studio supplies the interface between the simulator environment and the DSP, it is possible to use the debugging capability of Code Composer to develop and evaluate the SDR algorithm. With the advent of Code Composer Studio release 2.0, it is also possible to simulate the algorithm on a variety of DSP simulators that are provided with Code Composer as well as the actual target DSP. Finally, the RTDX channel transfers data to and from the JTAG emulator interface during the idle time of the DSP CPU. Therefore, it is also possible to accurately measure the execution time of the algorithm, since the transfer of data via the RTDX channel does not add significant overhead to the DSP algorithm being considered.

The advantage of a drag and drop block diagram approach offered by SystemView or (more recently) Matlab’s Simulink, provides the ability to quickly vary inputs to the algorithm as well as model external components of the SDR system such as RF and analog components as well as the RF environment of the channel itself.

Finally, both SystemView and Matlab offer the capability of developing bit-true DSP algorithms from the block diagrams developed in their respective simulation environments. Once completed, these algorithms can be converted to optimized C code that can be compiled and executed on Texas Instruments DSPs. Thus it is possible to significantly reduce the SDR algorithm development time while automatically providing a benchmark simulation by which the performance of the DSP algorithm can be compared.

Author’s Note

Since the preparation of this article, the simulation has been expanded to include a complete model of the analog receiver using SystemView’s RF and Communication token library. This model simulates the frequency plan, power budget, noise figure and 3rd order spurious response.

Readers that are interested in further details may contact the author: Robert G. Davenport, MC^2 Technology Group, LLC, e-mail: rgdavenport@rochester.rr.com.