Developing a GSM Modem on a DSP/FPGA Architecture

Using System Generator and Simulink, you can create a seamless simulation-to-implementation FPGA design flow.

by Louis Belanger
Product Development Manager
Lyrtech Signal Processing, Inc.
louis.belanger@lyrtech.com

GSM (Global System for Mobile) is the most widely-used cellular phone technology. Having begun mostly as a European standard, it has spread throughout the world to become ubiquitous.

GSM was one of the first digital cellular systems, and as such, represented another level of magnitude in terms of complexity, with support for growth features such as GPRS (General Packet Radio Service), which provides data capabilities to GSM phones. In this context, designing a GSM system is a complex task and could benefit from advanced design flow techniques where initial system simulation phases can be seamlessly carried over to the implementation phases.

In this article, we'll describe such a design flow for GSM development, starting from research and model simulation/implementation in The MathWorks MATLAB™ to FPGA implementation through simulation phases in The MathWorks Simulink™ and Xilinx® System Generator.

Project Context

Our implementation is in the general context of wireless application development examples to showcase our DSP/FPGA platforms. A previous project, the implementation of a SSB (single side band) AM radio (also with The MathWorks) featured a simpler analog radio and was showcased at Xilinx Programmable World 2003.

We wanted to implement a much more complex radio, and so we selected the GSM digital cellular standard. In doing so, we are getting closer to our goal to design ever-more-complex wireless systems for our customers.

We wanted to have a simplified system that still operates as a GSM system, while demonstrating a model-based (also known as system-level) design flow. The target platform is our SignalMaster DSP-FPGA, with high-speed sampling boards to sample the intermediate frequency (IF) coming from a special-purpose radio frequency (RF) front-end. We performed IF processing in the Xilinx Virtex-II™ FPGA and developed the design using System Generator.

In a complementary manner, baseband processing implementation occurs in the DSP, using a similar Simulink design flow with the The MathWorks Real-Time Workshop™ C-code generator, Embedded Target for TI DSP toolbox, and LYRtech's GSM DSP libraries.

We designed protocol-oriented transactions using The MathWorks Stateflow™, a

tool that allows the graphical design of states and transitions, as well as the production of associated code. This event-driven code can run on the DSP itself or on a companion RISC processor.

GSM Processing

Figure 1 depicts a GSM physical channel. There are 124 200-kHz channels that are frequency multiplexed in a 25-MHz-wide RF spectrum, one for each downlink and uplink path. Figure 1 also shows in more detail how the "bursts" of each GSM channel are constructed.

Basically, each burst is part of an 8-slot time division multiplex frame, forming a 200-kHz-wide spectrum. Each one has tail bits and an extended guard interval to avoid interference, as long as the mobile station (MS) is within 35 km of the base station (BS). Some fixed training-bit sequences allow synchronization between the MS and the BS.

Using the GSM as a design example corresponds very well to our platform's segmented architecture. The main uplink physical layer elements of a GSM speech and data transmission chain are shown in Figure 2. Figure 2 also illustrates how to partition the processing.

We performed IF processing on the FPGA with functions such as polyphase DDC (digital down converter), DDS (direct digital synthesis), and GMSK (Gaussian Minimum Shift Keying) modulation. DSP-based baseband processing can tackle the tasks of encoding, encrypting, and interleaving, as well as burst building functions. Finally, communication protocol handling occurs at the RISC processor level (or in the DSP for simplified protocols, such as in our case).

Simulink DSP Model

The DSP section of the GSM model contains the following elements:

- All higher level protocol layers
- Baseband processing modules
- Data transfer protocol for mapping data onto 32-bit frames, before sending to the FPGA through the parallel bus interface

Figure 3 displays the DSP model, which contains the baseband processing block and Stateflow diagrams. This figure shows a combination of The MathWorks' MATLAB off-the-shelf functions and target-specific library blocks.

This is very typical of a model-based design flow, in which target-specific block performance is compared with pure simulation blocks. At target compilation time, the associated DSP library of these last blocks can build the DSP code, providing very efficient code generation and performance.

FPGA Processing Model-Based Design

Figure 4 illustrates the main FPGA-based Simulink model for the base station. On the transmit side, the FPGA receives 148-bit GSM frames from the DSP as input, modulates them to get a GMSK burst, and then frequency-shifts the resulting signal in the 70 MHz IF band through SSB modulation, using the Xilinx direct digital synthesizer (DDS) block.

Two FDM transmit channels are simulated in this model. These two signals are then mixed together on the same physical channel and sent to the digital-to-analog converter (DAC); that block is located in the Signal I/O and Mixer subsystem.

On the receiver side, signals feeding the FPGA come from the analog-to-digital converter (ADC). Because the digitized signal is 25 MHz wide and can contain as many as 124 GSM channels, channel selection is required. This is performed by a

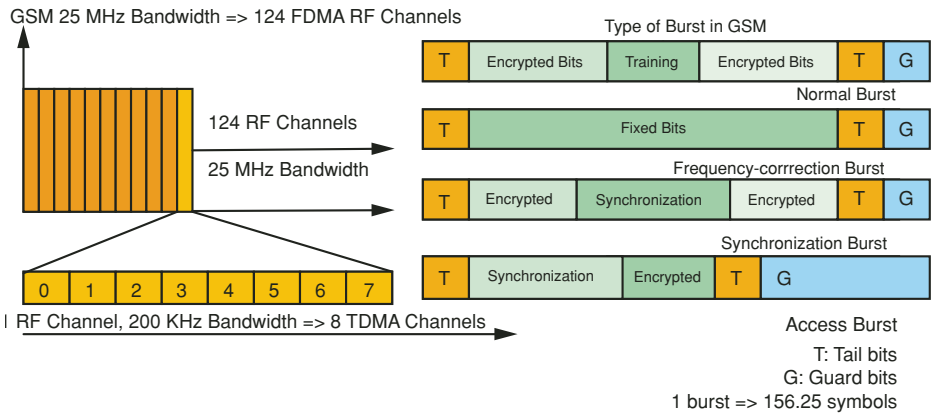

Figure 1 – GSM FDM, TDM, and burst structure

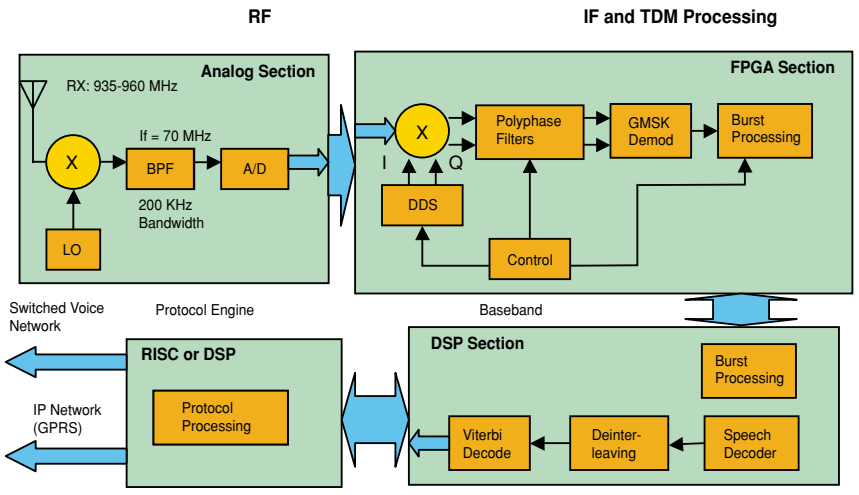

Figure 2 – Partitioning of the GSM processing between the FPGA, the DSP, and a RISC processor

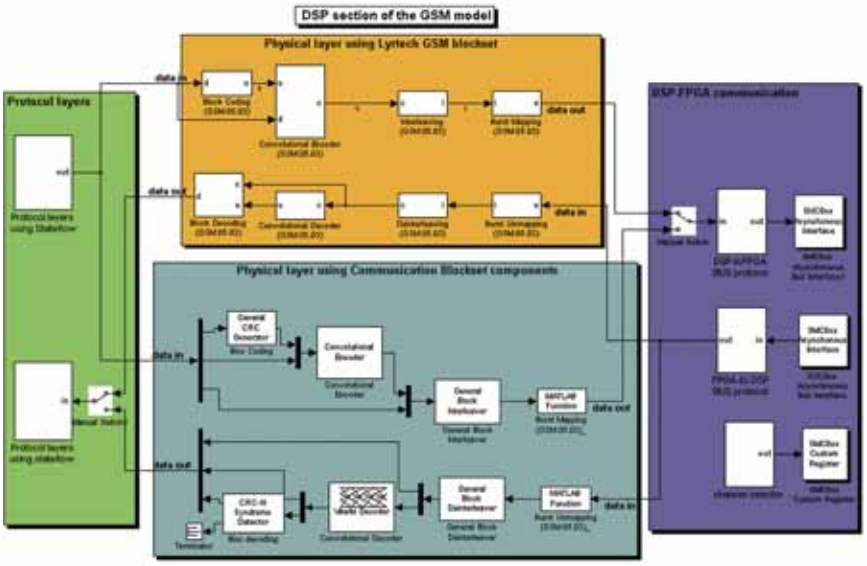

Figure 3 – DSP model of baseband GSM processing with comparative host/target blocksets

DDS in the IF demodulator subsystem; that frequency is dictated by a value coming from the DSP. The baseband signal can now be GMSK-demodulated and sent for further processing.

This model runs in one of three different ways:

- Normal mode (simulation)
- Co-simulation mode (hardware-in-the-loop)
- Real-time mode (100% hardware)

In normal mode, the data comes from the block located in the DSP section. This block contains the DSP functions shown in Figure 2, in order to provide frames to the transmitter (the same applies for the receiver side).

Simulink performs all processing during simulation. The gateway blocks have no effect, except for converting the signal from one format to another (such as from double to fixed-point). The Signal I/O and Mixer subsystem then produces a loopback of the IF signal and adds white Gaussian noise to it.

The co-simulation mode basically performs in the same way as the normal mode, except that the processing function between the gateways will be executed in the FPGA as a hardware-in-the-loop simulation.

In real-time mode, all the blocks outside the gateways are ignored. These gateways establish the communication between the different hardware entities. In the case of our GSM model, the frames travel from DSP to FPGA and vice-versa through the parallel 32-bit data bus.

The IF signal now travels through the DAC and can either be in loopback mode, if the output of the DAC is connected to the input of the ADC, or fed to the front-end to produce an RF signal.

Co-Simulation Benefits

We designed the first version using only standard communication and DSP blocksets from Simulink, running in double precision from start to end. As a second step, we gradually replaced Simulink blocks with ones from Xilinx, and tested the model in normal mode, which resulted in hybrid models and simulations that were easier to debug.

After producing the Xilinx model, we could test it in co-simulation to verify that hardware computation was as expected. As a final step, we targeted the whole process to hardware.

Demodulator Subsystem Complexities

The subsystem presented in Figure 5 is the IF-to-baseband demodulator, which is the most complex part of the design. Before

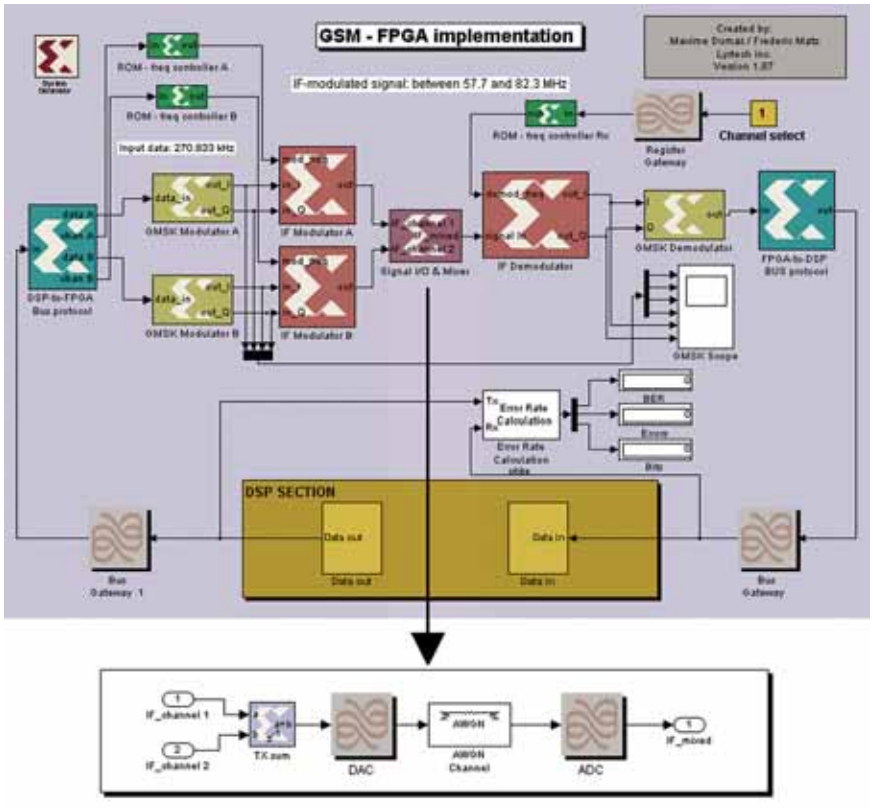

Figure 4 – FPGA model/IF processing of GSM

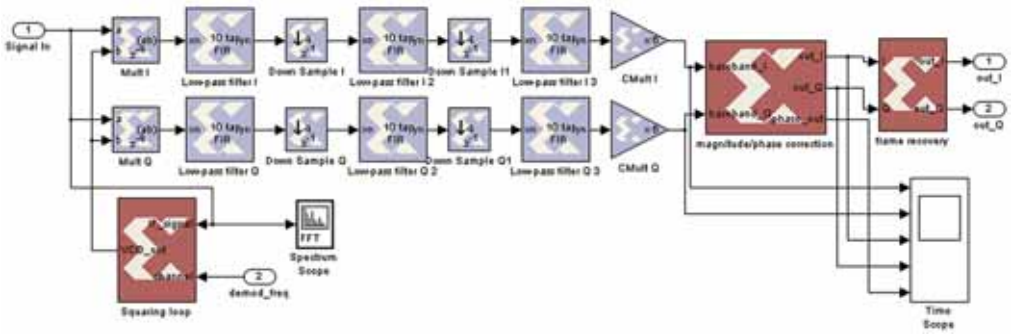

Figure 5 – IF FPGA model/demodulator subsystem

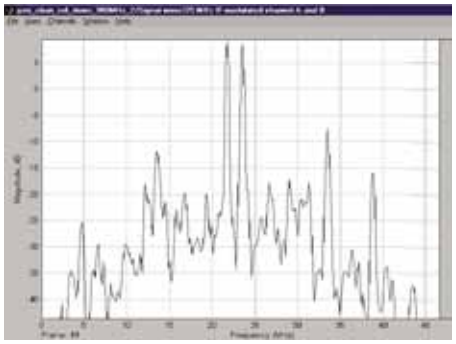

Figure 6 – Spectrum scope displaying the two GSM FDMA channels

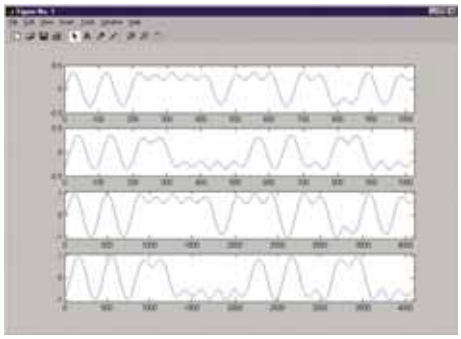

Figure 7 – Time scope displaying I/Q signals before and after the magnitude/phase correction operation

the GMSK demodulator receives the signal, the receiver has more to do than just blindly shift and down-sample the signal from IF to baseband. It must compensate for the effects of the channel on the signal, which means it has to perform the following:

- Carrier frequency recovery
- Carrier phase recovery
- Amplitude adjustment
- Timing recovery

We created low-pass filters using the digital filter design block from the MATLAB DSP blockset, which were later replaced by the Xilinx FIR filters that use the same taps generated by the Simulink block. Once the demodulator was functional for ideal signals, we added correction blocks to cope with non-ideal signals.

You can see in Figure 5 that a squaring loop deals with carrier recovery, while the magnitude/phase correction block takes care of the remaining amplitude and phase errors with trigonometric properties of a quadrature signal. We performed a cross-correlation with the expected training sequence to recover the timing and send a complete frame to the GMSK demodulator.

Hybrid FPGA/Simulink modeling was very useful in the development of this subsystem, because it was possible to visualize the signals at every step of the processing, both in time and frequency. Figure 6 displays the spectra of the two FDMA channels before demodulation and channel selection, while Figure 7 shows the waveform obtained before and after the magnitude/phase correction block.

FPGA Resource Estimation

Table 1 shows the resources used in the Virtex-II FPGA.

The IF-baseband demodulation is based mainly on a three-stage decimation and filtering applied to both I and Q signals, each using three 10-tap FIR filters.

Basically, the implementation of the GMSK demodulator brings together three major components: the phase recovery module, the timing recovery module, and the MLSE (maximum likelihood sequence estimation). The phase recovery module uses some dividing and square-root operators, which are costly to implement. The timing recovery is based mostly on the correlation of large data sequences that demand many embedded multipliers, and also some large data buffers made out of block RAM

The implementation of an MLSE comprises a modified version of the Viterbi algorithm and demands considerable resources. This full-feature demodulator can be optimized, simplified, or targeted at an ASIC, but as a first-pass iteration, it provides a good estimation of the resources needed for its implementation.

	Slices	Total Slices (%) Virtex II XCV3000
GMSK modulator	190	1.33 %
IF-baseband modulation/demodulation	6,567	45.80 %
GMSK demodulator	4,775	33.30 %
BUS gateway and protocol	382	2.67 %
TOTAL :	**11,914**	**83.10 %**

Table 1 – Resources used in the FPGA

Conclusion

A model-based design approach in the development of a complex wireless application for a DSP/FPGA architecture is very effective for thoroughly testing a design while implementing it for target hardware. Nonetheless, the use of FPGA cores and DSP libraries also allows the implementation to be quite efficient, even with a high-level design approach. Combined with flexible platforms such as the SignalMaster, these tools and approaches really help today's designers tackle the difficult challenges of designing state-of-the art wireless systems.

For additional information on this project and our SignalMaster line of DSP/FPGA development platforms, visit *www.lyrtech.com.*