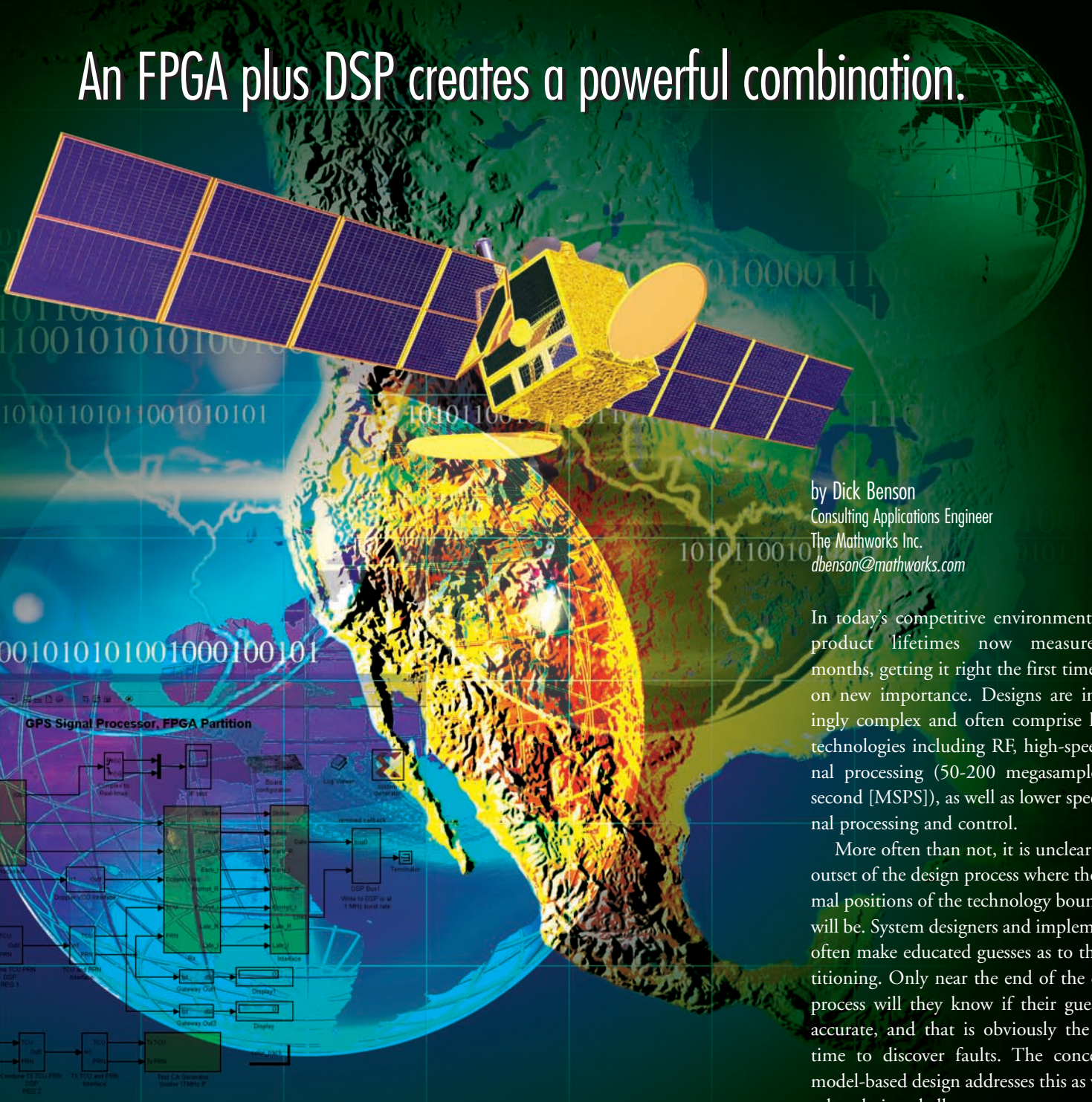
The Design and Implementation of a GPS Receiver Channel

An FPGA plus DSP creates a powerful combination.



by Dick Benson
Consulting Applications Engineer
The Mathworks Inc.
dbenson@mathworks.com

In today's competitive environment, with product lifetimes now measured in months, getting it right the first time takes on new importance. Designs are increasingly complex and often comprise hybrid technologies including RF, high-speed signal processing (50-200 megasamples per second [MSPS]), as well as lower speed signal processing and control.

More often than not, it is unclear at the outset of the design process where the optimal positions of the technology boundaries will be. System designers and implementers often make educated guesses as to the partitioning. Only near the end of the design process will they know if their guess was accurate, and that is obviously the worst time to discover faults. The concept of model-based design addresses this as well as other design challenges.

Model-Based Design

Having one set of cost-effective integrated tools that you can use to design, verify, partition, and automatically generate code for both FPGAs and DSPs is now a reality. The Mathworks calls this process "model-based design."

The concept is quite simple. First, you create a functional implementation independent model of the system. This is an "executable specification," a model that forms the basis of all that is to follow. Once you have verified the model to achieve your system objectives, you can incorporate further detail, such as adding fixed-point effects, RF/ADC non-idealities, and partitioning the design between high-speed fixed-point hardware (an FPGA) and lower speed hardware (a DSP). At every step of the process, you verify that the model achieves the performance goals. The final step is to use the automatic code generation capability to flawlessly implement the model on hardware.

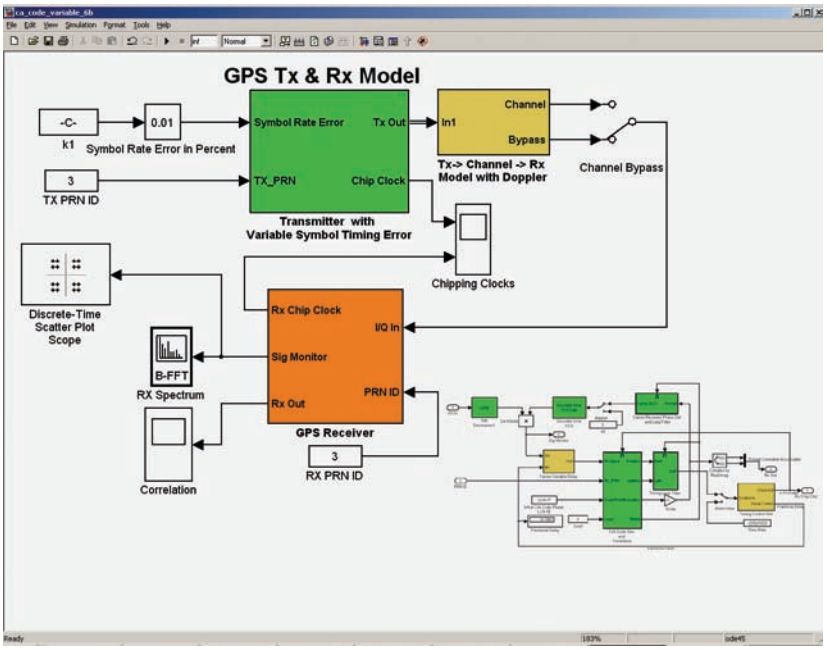


Figure 1 – A top-level view of a GPS system, including a transmitter with timing error, channel model with Doppler, and a receiver with timing and Doppler de-rotation loops. The model contains numerous levels of hierarchy. A glimpse down one level into the receiver is shown in the lower right.

GPS Receiver

The GPS system has been fully operational with 24 satellites in its constellation since 1994. It is used by millions of people, both civilian and military, every day. The fundamental concept of using code-division multiple access (CDMA) for time-delay measurement (yielding range) while allowing all satellites to share the same carrier frequency (1.57542 GHz for civilian access) has not changed in the past 30 years.

Each GPS satellite uses a unique 1,023-chip orthogonal code (Gold code) to spread the low-speed binary phase shift keying (BPSK, 20 ms per bit) navigation data bitstream. The chipping clock rate is 1.023 MHz, and therefore the sequence of 1,023 chips repeats every millisecond.

The GPS receiver generates a local copy of the same Gold code, which is then cross-correlated with the incoming signal. When the receiver code phase aligns with the incoming signal code phase, there is a +30 dB improvement in the SNR (10*log10(1023)), and the BPSK navigation bitstream can then be easily detected. Roughly speaking, you can use the local code time offset, where the correlation is maximized to estimate the time difference between the received signal and the signal transmitted by the satellite. The range is directly proportional to time; therefore, with four satellite ranges, and knowing the positions of the satellites, you can navigate.

Like all CDMA systems, the design of a GPS receiver presents some formidable challenges. The satellites are traveling at close to 7,000 km per hour, giving rise to a Doppler shift ranging from -6000 to +6000 Hz. They are 20,200 km away, running low-power (50W) transmitters. Signals at the terrestrial receiver input are typically 20 dB or more below the noise level. And the local receiver clock is never in step with the ultra-precise Cesium clocks in the satellites, so Doppler and symbol timing recovery loops need to be implemented as well as acquisition logic.

An Executable Specification

Figure 1 shows a top-level view of a Simulink model. It contains the transmitter, channel, receiver, and measurement visualization subsystems. This model has numerous levels of hierarchy – a glimpse under the hood of the receiver is shown in the inset. The transmitter model allows timing errors to be introduced while the channel model includes Doppler shift. These impairments test the receiver timing recovery and carrier tracking loops.

Once the simulation meets the required performance goals, it becomes an executable specification. In theory, the model implements the GPS physical layer according to written specifications that you can find in the public domain.

Verify the Model and Partition

Studies have shown that defects are most often introduced at the beginning of a design. Before adding more detail to the model, it is prudent to verify that it truly implements a GPS receiver. The MATLAB language is becoming increasingly popular in test and measurement applications. The Instrument Control Toolbox option for MATLAB can communicate with virtually any instrument that has a hardware interface. Beyond this, several test and measurement vendors have integrated MATLAB into their instruments. Anritsu is one such company; their Signature spectrum analyzer can capture data into MATLAB with a single mouse click.

An antenna and low-noise pre-amplifier were connected to the Signature analyzer, and tuned to 1.57542 GHz. We recorded approximately one second of I/Q format data, which was then available in the MATLAB workspace. Note that since the satellite signals are more than 20 dB below the noise, it is not immediately obvious that a useable data set has been captured. We used a separate Simulink model implementing a simplified GPS receiver (no tracking loops) to confirm that satellite signals were present in the data. The transmitter portion of the original model was then replaced with a Simulink library block to provide actual satellite data for testing the receiver model.

Next, the model is partitioned into a portion that will reside in the FPGA and a portion that will reside in a floating-point DSP.

The incoming I/Q data at the 8 MSPS rate is first passed through a root-raised cosine FIR filter. Naturally, this higher speed processing is best suited for the FPGA. The filtered signal is then down-sampled by a factor of two, and after the Doppler de-rotation, feeds three cross-correlators: early, prompt, and late, which refer to the local de-spreading code phase driving the respective correlators. The numerically controlled oscillators for both the Doppler and the local de-spreading code are also in the FPGA partition. Because the de-spreading sequence repeats every millisecond, the outputs of the three correlators are only of interest at this one-millisecond rate, which is easily handled by a DSP.

After the receiver model is working using floating-point arithmetic, the next step is to define the fixed-point attributes that will be required for the FPGA partition.

Simulink models can accommodate arbitrary precision fixed-point representations of signals. The FPGA partition includes these fixed-point constraints. As before, numerous levels of hierarchy exist in this model, and Figure 2 represents the top level. Notice that the partitioning reveals a feedback control system between the DSP and the FPGA.

In review, the 8 MHz I/Q satellite signal input is processed by the FPGA producing low-rate (1 kHz) correlator outputs, which are then processed by the DSP. Using these signals, the DSP in turn implements the proportional-integral-derivative controllers for both the Doppler and timing recovery loops. The two controller outputs are fed back into the FPGA. The real captured satellite data is again used as a source to test the partitioning and the chosen fixed-point data constraints. Figure 2 shows the satellite data source (picture), the FPGA partition (yellow), and the DSP partition (green).

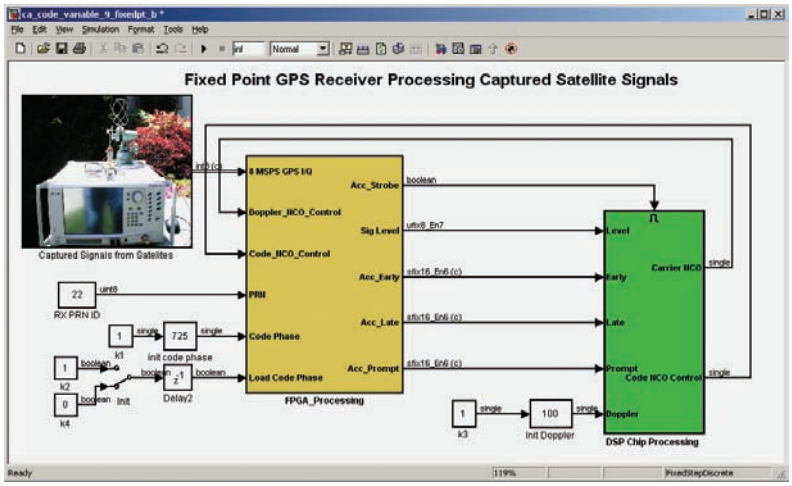


Figure 2 – The receiver portion of the model has now been partitioned with the high-speed fixed-point portion in yellow, and the lower speed single precision floating-point portion in green. RF from actual GPS satellites is captured with a spectrum analyzer. This is now the data source for verifying the model with real-world data.

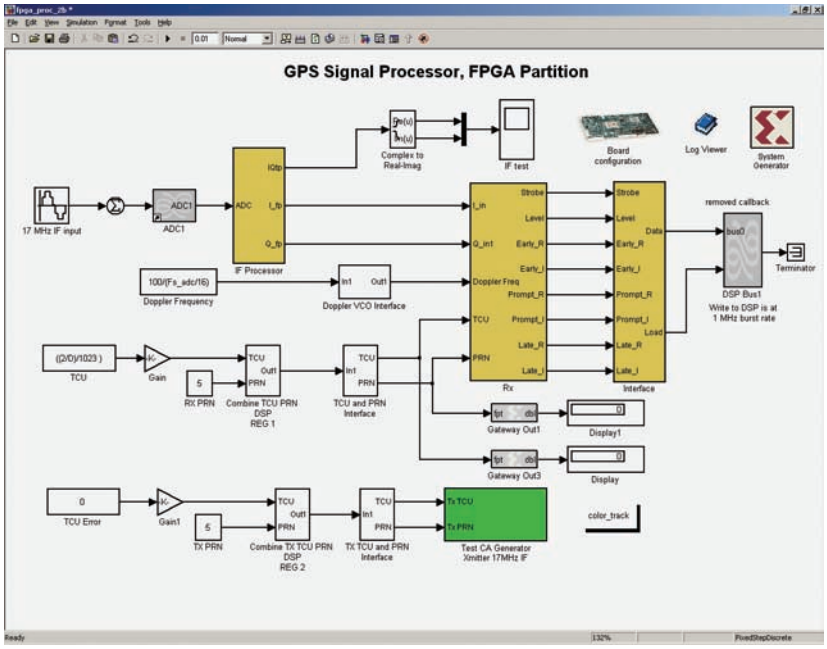


Figure 3 – The fixed-point partition in Figure 2 is now implemented using blocks from the Xilinx System Generator library. A digital down converter (IF processor) has been added along with a 64 MSPS ADC to form a front end for the base-band receiver subsystem. The correlation data from the receiver is time-division multiplexed and buffered with a FIFO before being fed to the DSP partition. All of the VHDL (or Verilog) required to implement this is automatically generated using Xilinx System Generator.

Implement the FPGA Partition

It is now relatively easy to transition the fixed-point receiver subsystem in Figure 2 to one using blocks from the Xilinx® System Generator for DSP library. The transition to

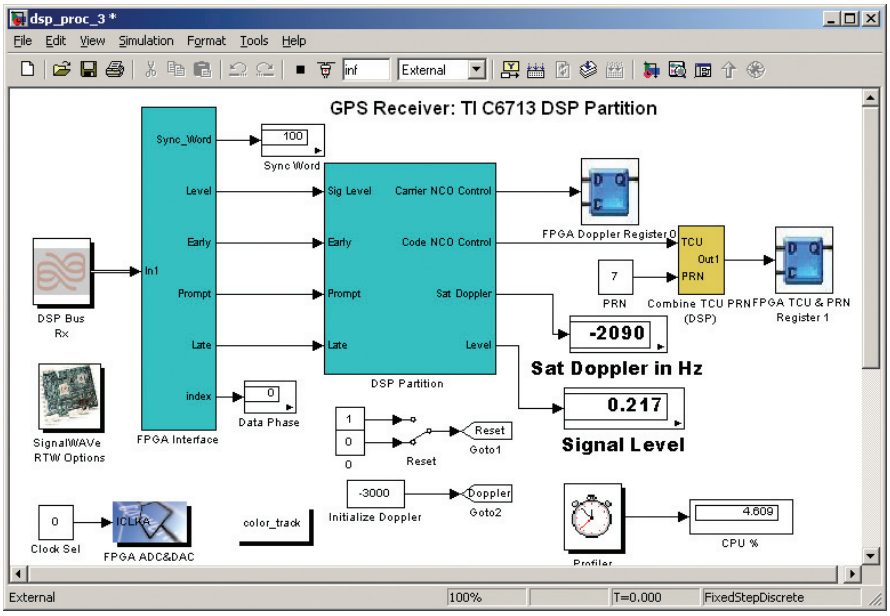


Figure 4 – The floating-point portion of the receiver is implemented using a TI C6713 DSP. The data from the FPGA is de-multiplexed into the signals required to implement both acquisition and tracking of the CDMA satellite signal. The control signals back to the FPGA pass through hardware "gateways" that are specific to the Lyrtech SignalWave hardware. The C code to implement this partition is automatically generated using The Mathworks' Real-Time Workshop. Once the bitstream and binary are loaded to the hardware, the block diagram becomes a user interface, allowing you to change parameters on the fly and have dynamically updating readouts, including time-history scopes.

the FPGA is easy if Simulink blocks that functionally match those in the System Generator library were used.

Figure 3 shows the FPGA portion of the design. Hardware-specific gateway blocks pass signals between the FPGA and the DSP. The output signals from the FPGA include three complex correlation signals (early, prompt, late), a signal-level estimate for AGC, and a synch word. These eight values are time-division multiplexed into a single data stream and fed through a 32-bit gateway block back to the DSP. Control signals for the timing and Doppler tracking loops come through 32-bit gateway blocks to the FPGA, along with other ancillary control signals such as satellite selection.

When running the hardware in real time, the signal input to this GPS receiver comes from an analog front-end down converter with an IF of 17 MHz, not a spectrum analyzer. Therefore, a digital down converter (DDC) is also needed in the FPGA. The DDC takes this 17 MHz real-band pass signal sampled at 64 MSPS and translates it to a base-band I/Q signal running at 8 MSPS.

The high-level signal processing functions (DDS, CIC, FIR, FIFO, TDM) from the Xilinx System Generator library make it easy to implement this portion of the design. Once the simulation is verified, a click of the mouse automatically generates the circa 350 VHDL files (750 kB of ASCII) required to implement the design. After this point we are in the standard Xilinx ISE™ design flow.

Implement the DSP Partition

The top-level view of the DSP portion is shown in Figure 4. The time-division multiplexed signals from the FPGA arrive through the gray gateway block on the left of the model. They are then de-multiplexed into the early/prompt/late correlation signals and level required by the timing recovery and Doppler controllers implemented in the DSP partition. The task of acquiring the GPS is implemented in the DSP using Simulink's event-driven option, StateFlow. Operations such as square root and arc tangent are required. Although these operations are possible with a CORDIC in the FPGA, they are even easier to do in the floating-point DSP.

Because the signals from the FPGA arrive every millisecond, the processing is light duty for the DSP. The numerical readout in Figure 4 indicates that only 4.6% of the available DSP horsepower is being consumed. That said, it should be noted that this example implements one "channel" of a GPS receiver in that only one satellite can be received. A typical GPS receiver incorporates six to ten channels and the loading of the DSP will increase in proportion to the number of channels.

The C code for the DSP is automatically generated using the Real-Time Workshop option in the Simulink environment. Once this is complete, a click of the mouse downloads both the bitstream for the FPGA and the binary for the DSP to the Lyrtech SignalWave hardware. The SignalWave has a Virtex™-II Pro XC2V3000 FPGA 64 MSPS ADC and DAC, as well as audio and video CODECs.

When the real-time processing is started in the hardware, the Simulink block diagram then becomes a GUI that allows you to seamlessly interact with the processing. Scopes and numerical readouts are the primary real-time display options. You can also change the state of switches, the values of constants, and multiplier gains to interact with the design without stopping or introducing gaps in the real-time processing.

Conclusion

This operational GPS receiver was developed using tools from Xilinx, The Mathworks, and Lyrtech. Not a single line of code was hand-written for either the FPGA or the DSP. It took approximately six weeks to create – from scratch – a receiver producing the navigation data bitstream from RF satellite input signals.

This model-based design example has been privately presented to several GPS design groups. Feedback indicates that accomplishing what we have shown in this article typically takes these designers more than a year.

If you are designing and implementing complex signal processing systems for real-time hardware, you cannot afford to be without these tools.

For more information, visit *www.mathworks.com*, *www.xilinx.com/systemgenerator_dsp*, and *www.lyrtech.com*.