Multiprocessor Networks

Hendri Veldman
3L Ltd

This article examines ways in which tool flows for multiprocessor systems can give flexibility to engineers for executing logic in different hardware and in different processor configurations.

Critical processing designs increasingly combine GPP, DSP and FPGA processing resources; Software Defined Radio is a key example. The GPP handles the overall system control, the DSP the high speed signal processing, and the FPGA the custom elements, often related to up-down conversions and filters. Traditionally, this requires three distinct design tools and creates hard boundaries between the processors.

Embedded design tools have invariably been aimed at single-processor systems. While these tools are well-designed for this environment, they offer little help when targeting multiprocessor systems. Working with multiple processors can be a complex and time-consuming endeavour: you need to deal with many tedious housekeeping operations such as loading the system, starting, synchronizing, and communicating between the various processing elements.

Newer techniques consider these three processor classes to be increasingly interoperable and in some cases exchangeable, blurring the lines between hardware and software deployment and keeping options open for as long as possible in the development process.

Channel-based Multiprocessor System Design

3L Diamond provides an abstraction layer that allows you to defer many decisions until the final stages of building your application. It does this by using a simple model where you describe a system as a number of tasks which communicate via channels. In this model, systems are specified without considering the details of how you implement tasks or where you run them. For example, the correlator task below has one input and one output channel.
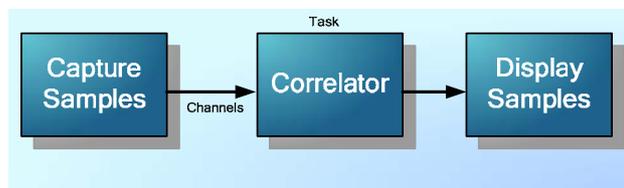
Once the system has been completely specified, you can experiment with where the tasks will run on the actual hardware. Multiple tasks can be made to run on the same processor and the software automatically takes care of loading and other maintenance issues, such as managing the communication between tasks.
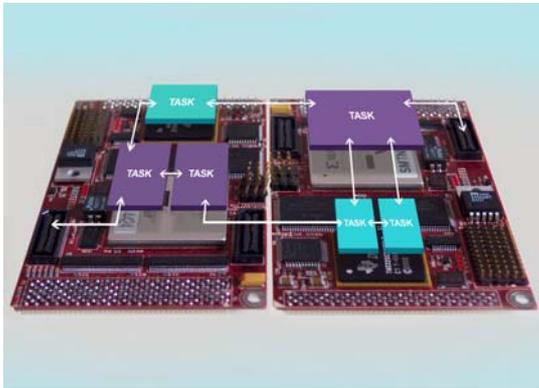

It is even possible to choose between equivalent implementations for the same task. For example, when a task is placed on a DSP, Diamond will automatically select the DSP version of the task. If you change your mind and place the same task on an FPGA, then the FPGA version is chosen.

The output from Diamond is a single application file that contains all the tasks for the complete system; as everything is self-contained, processors cannot be loaded incorrectly. The software also provides a host server that is used to load applications onto the target hardware. This server also displays debug information and provides the interface for standard I/O streams.

An Example With Six Tasks

The following example shows a system comprising six tasks. Some of these tasks represent windows on the host system that display graphics or prompt the user for input.
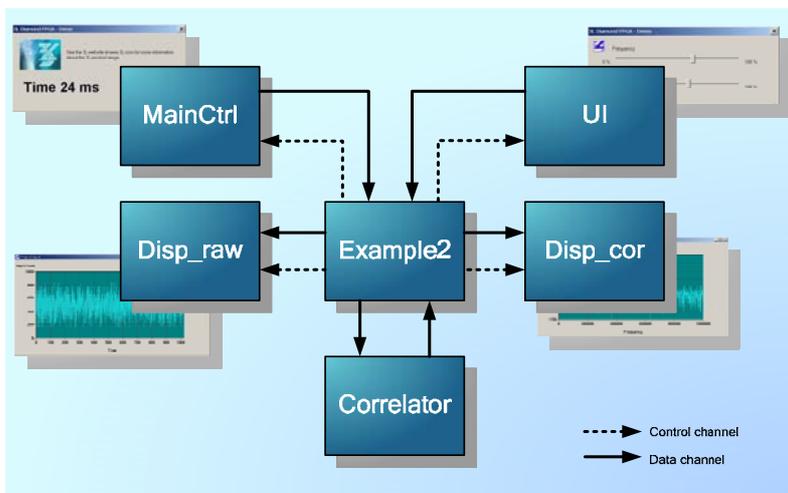
The *Example2* task creates a sine wave with some random noise added to it. This is then sent to a *Correlator* task, which calculates an autocorrelation. The results are returned to *Example2* which then sends them to a display task, *Disp_cor*. The original signal is also sent to another display task, *Disp_raw*, so that we can see both the original signal and the correlation. Diamond makes it possible to re-use tasks easily; in this example the two display tasks are simply two instances of the same task.

To test this prototype system we use a powerful DSP/FPGA development platform, the Sundance SMT395 module, which has a 1GHz C6416T and a Xilinx Virtex II Pro XC2VP30-6. Because Diamond forms an abstraction layer to the hardware, it would be trivial to change the actual hardware to completely different supported hardware if we so decided simply by changing the declared type of the processor.
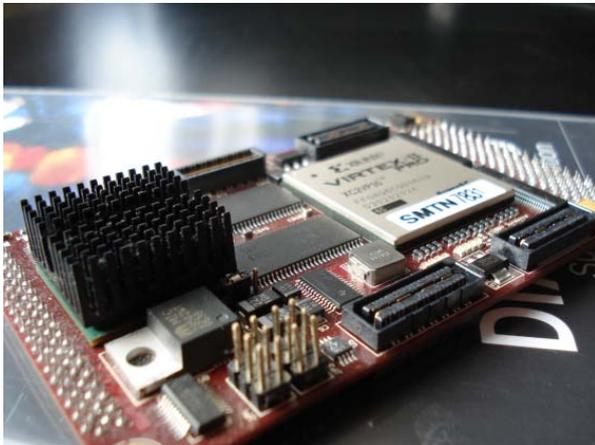

Accelerating Prototyping

Code executing on an FPGA can run considerably faster than the DSP equivalent, but the price for this is a greatly increased development time. The flexibility of Diamond allows system development to proceed with everything running on the DSP while FPGA versions of some tasks are being developed in parallel. The overall structure of the complete system can be validated, even though some parts of the system might not be running at full speed.

We will start by implementing the correlation example by running all the tasks on the DSP with nothing placed on the FPGA. To do this, we use the 3L Diamond IDE to specify the tasks and processors in the system. We can also specify how the tasks are connected, and where they should run, but in this example we are not interested in the connections between tasks.

The picture shows the six C tasks that make up our system, all placed on root, the DSP.
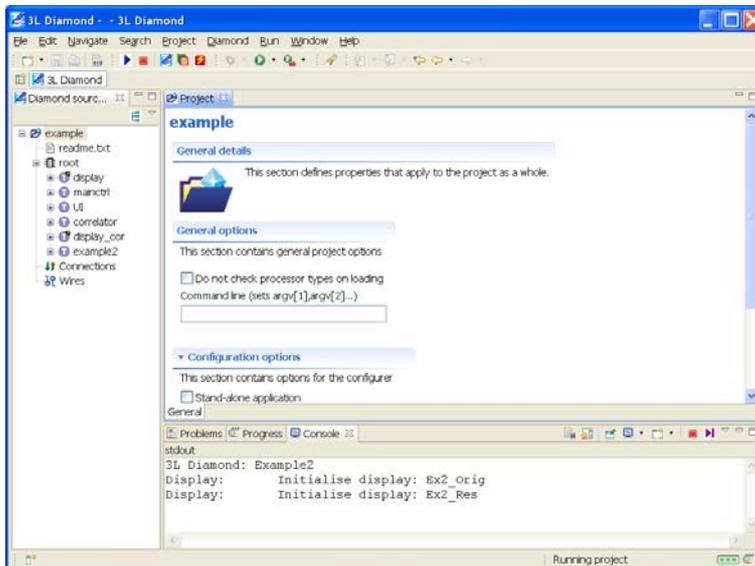
When we build this application, Diamond will call the Texas Instruments C compiler and linker to generate the tasks and will then combine them all into a single output file that contains everything needed to load the complete system. Now we can run the application and see it bring up the four windows shown below.
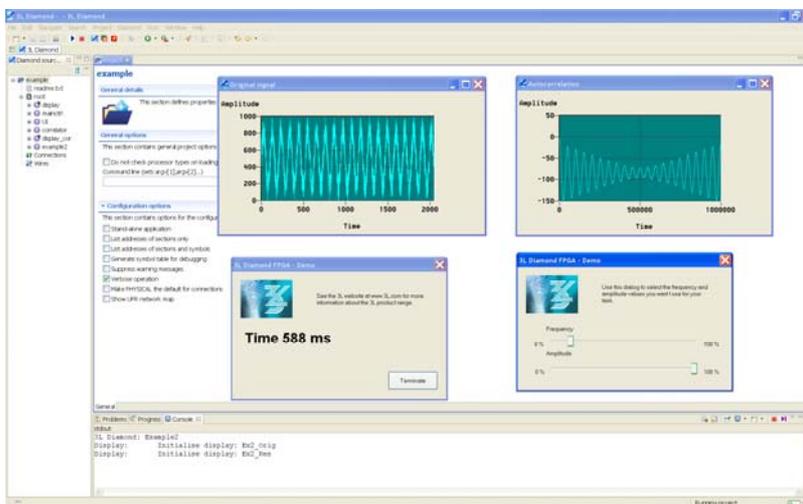

The time that is displayed shows how long it takes to send each block of data to the correlator task, to compute the correlation, and to send the results back. In this case that is 588ms.

Certain types of processing, such as correlations, are well-adapted to the highly parallel environment of an FPGA. We could develop an FPGA version of the correlator task directly in VHDL, but it is simpler to code the algorithm in C and use Impulse-C to target the FPGA.

Correlation on the FPGA

We start by creating an Impulse-C project, specifying a template that gives one input and one output. These form the ports of the correlator task that are used to connect it to other tasks with channels.
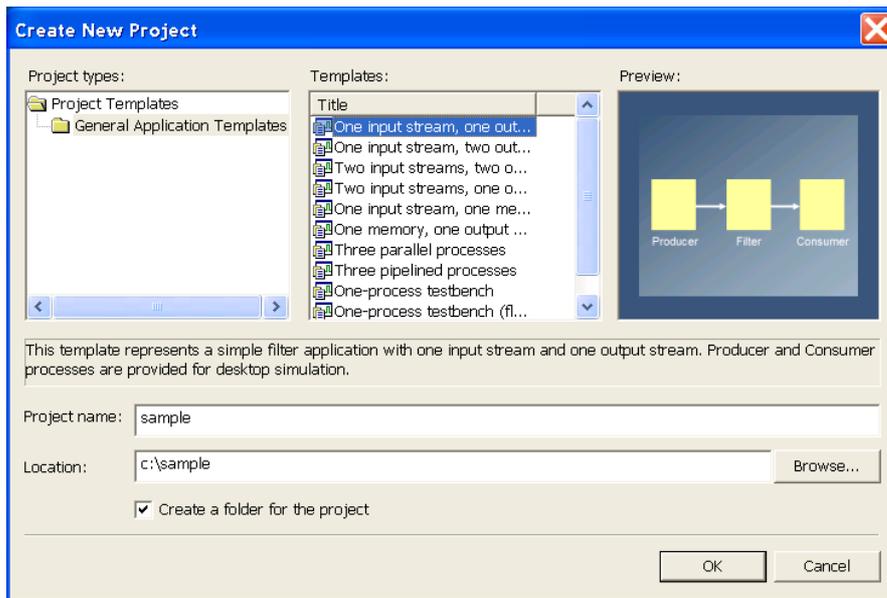

Next, we create a process named "Correlator".
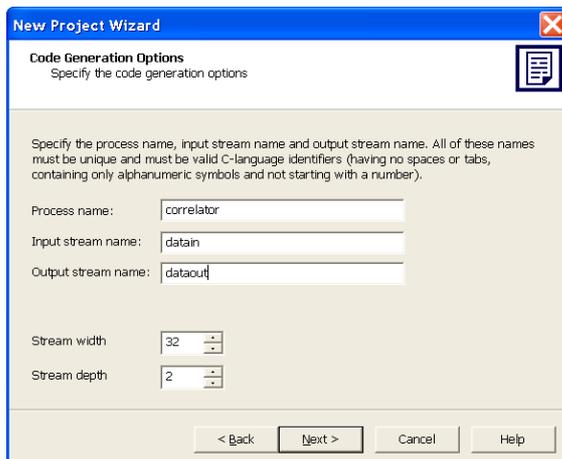
Since the correlator uses floating point, the "include floating point library" option is selected in the next set of options. As we want to be able to use the correlation task in the 3L Diamond environment, we also select "3L Diamond (VHDL)" as the platform support package.
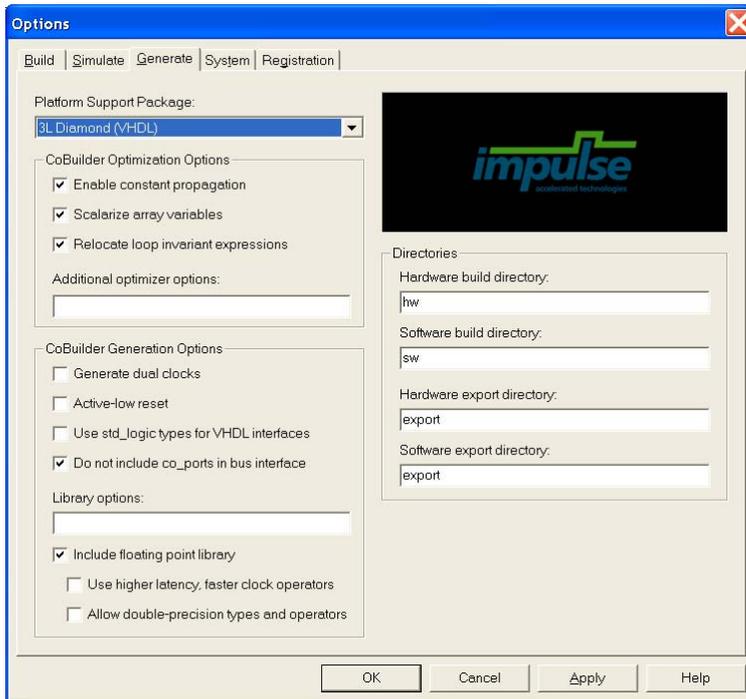
Next, the existing C code for the correlation task is imported into the Impulse-C environment.
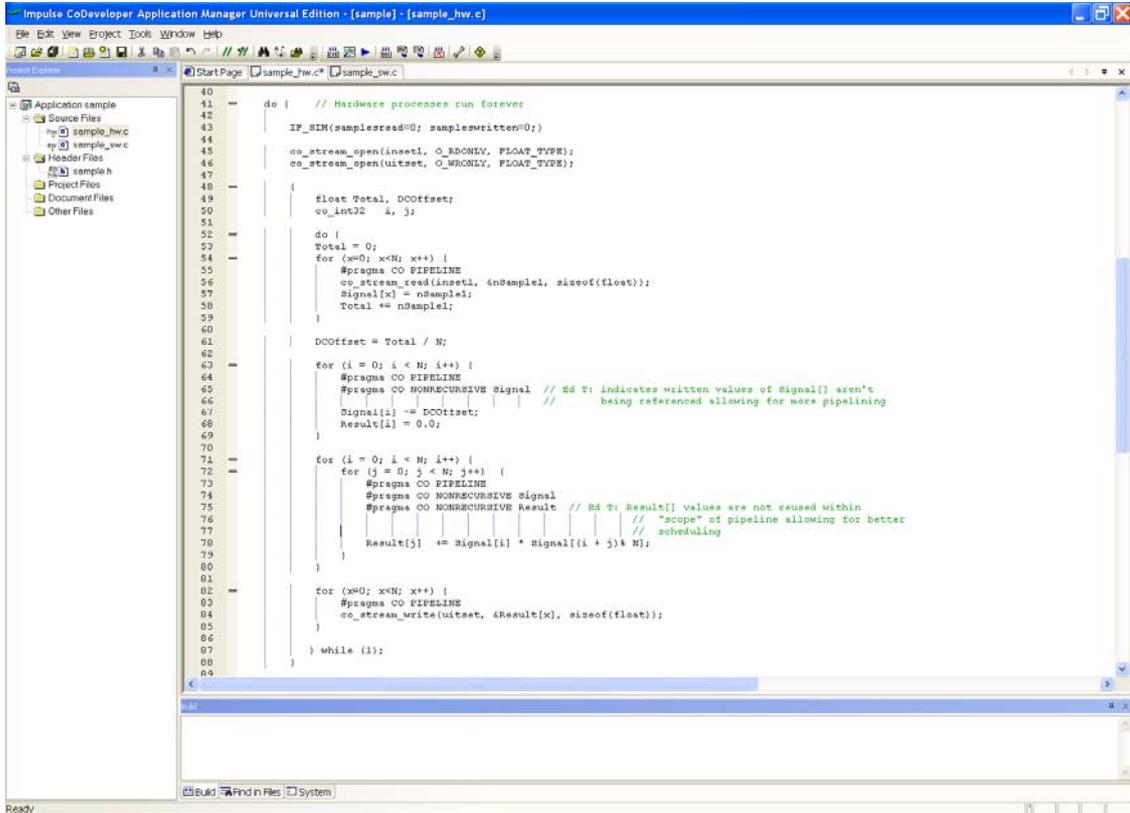
After several rapid iterations to experiment with the algorithm, a Diamond task is created by selecting "Generate HDL" from the Impulse-C tool. The resulting task can be placed on any Diamond-compatible hardware platform.

Placing the Correlator Task on an FPGA

Back in the Diamond IDE, we now declare the FPGA as a new processor "node".
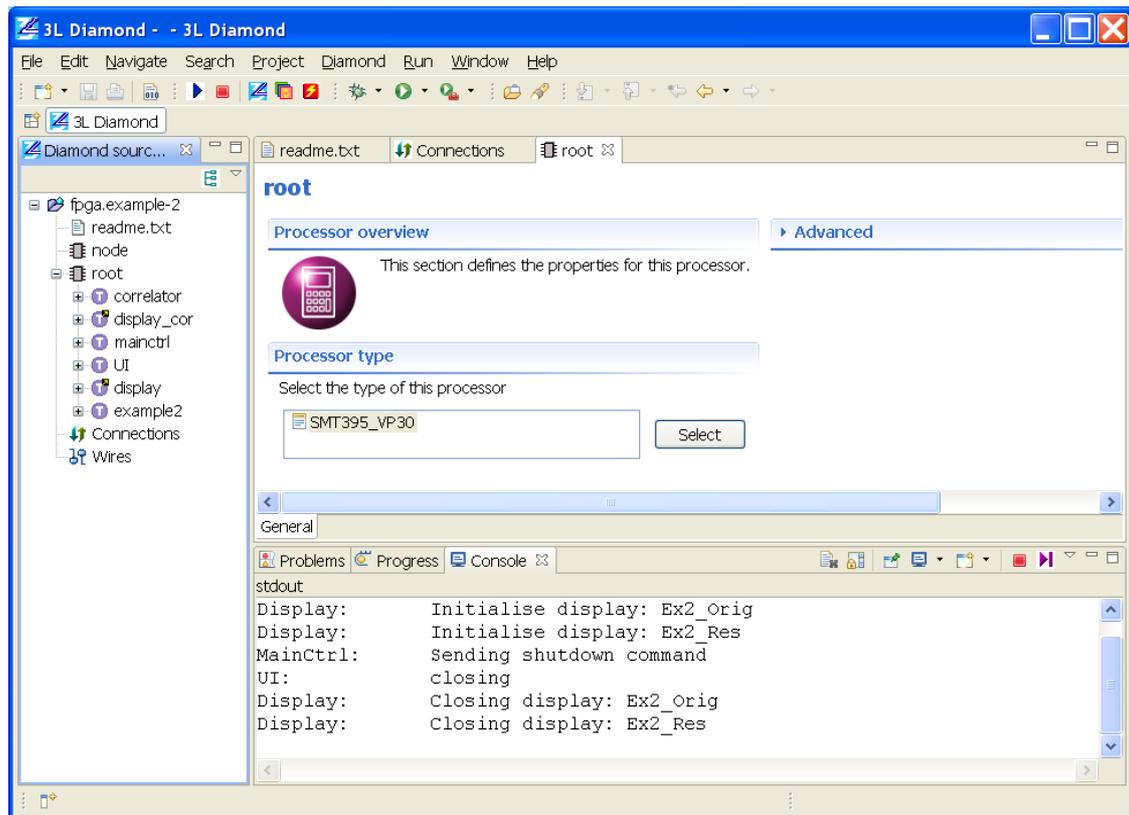
Next we drag the correlator task and drop it onto node, the FPGA. Nothing else needs to be changed since the system design remains unchanged: the tasks are still interconnected in exactly the same way they were before; we are merely changing where the tasks will run.

To select the Impulse-C generated version of the correlator task, we add the Impulse-C output, **task_sample_arch.fcd**, to the task's VHDL implementation.
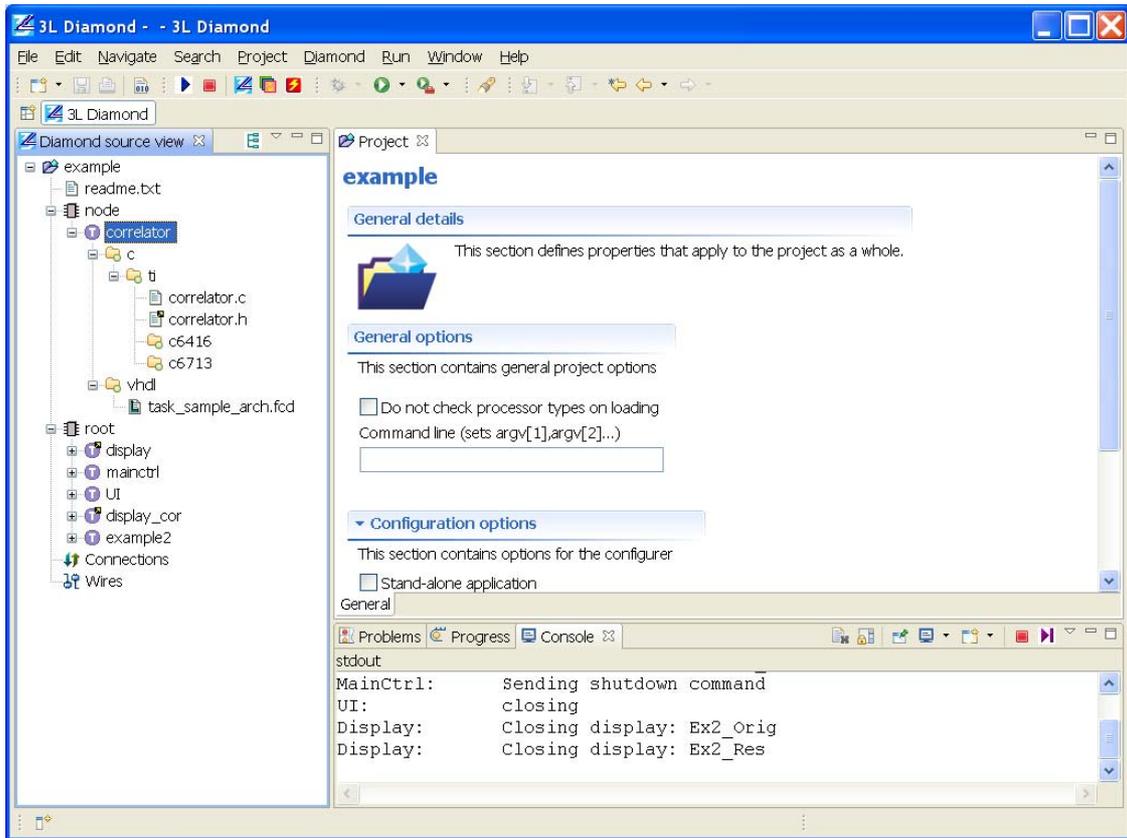
There are now two equivalent implementations of the same task: one for a DSP and one for an FPGA. Diamond automatically selects the implementation appropriate for the technology on which you place the task.

When we build this application, Diamond calls the Xilinx tools to build a bitstream for the FPGA and this is incorporated into the single output image. Running this now loads both the DSP and the FPGA, and gives the following output:
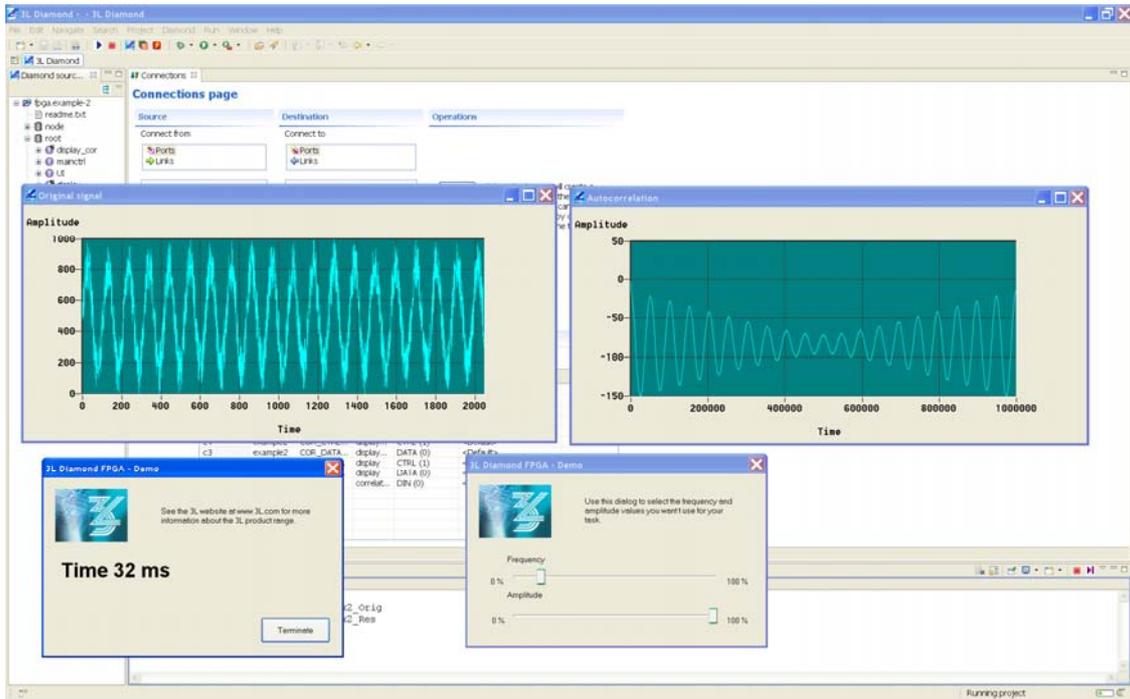
By moving the task onto an FPGA, the correlation time has reduced to 32ms, almost 20 times faster than before.

Summary

In this article we have shown how easy it is to target multiprocessor hardware by combining the 3L Diamond tools with the Impulse-C tools.   We illustrated how a correlation task can be implemented in Impulse-C, and how that task can then be run on real multiprocessor hardware.