Designing Multiprocessor Networks

Hendri Veldman
3L
hv@3L.com

Brian Durwood
Impulse Accelerated Technologies
brian.durwood@impulsec.com

Flemming Christensen
Sundance Microprocessor Technology
flemming.c@sundance.com

Abstract

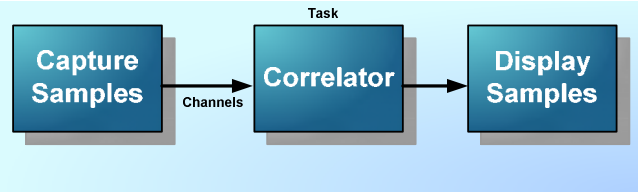
Critical processing designs increasingly combine GPP, DSP and FPGA resources with Software Defined Radio (SDR) being a key example. The GPP handles the overall system control; the DSP the high speed signal processing and the FPGA the custom elements that are often related to the up-down convert functions and the filters. To many users this would require three design tools and would create hard boundaries between the processors. Historically embedded design tools target single processor and whilst these tools are well designed for single processor environments, they do not work well when targeting multi-processor systems. Designing multiprocessor systems can be a complex and time-consuming task. You need to deal with things such as loading the system, starting, synchronization and communicating between the various processing elements.

Newer techniques are considering the three processor types as increasingly interoperable and in some cases exchangeable. These techniques are blurring the traditional lines between hardware and software deployment, keeping the lines fluid for much longer in the development process. This article looks at tool flows for multiprocessor systems and into the choices engineers have in executing logic in different hardware; logic that can be achieved in different stream configurations, in multiple processors and on single or multiple devices.

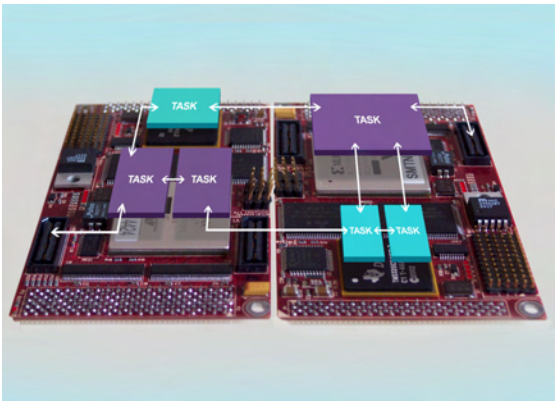
Channel-based multiprocessor system design

A simple model describes a system as a number of tasks that communicate via channels. By using this model, systems are specified without considering the details of where tasks will eventually be running on the hardware. An abstraction layer to the multiprocessor DSP and FPGA hardware is created that allows the designer to defer the decision of final processor target.

For example, with a correlator this would reduce the actual function to a channel.



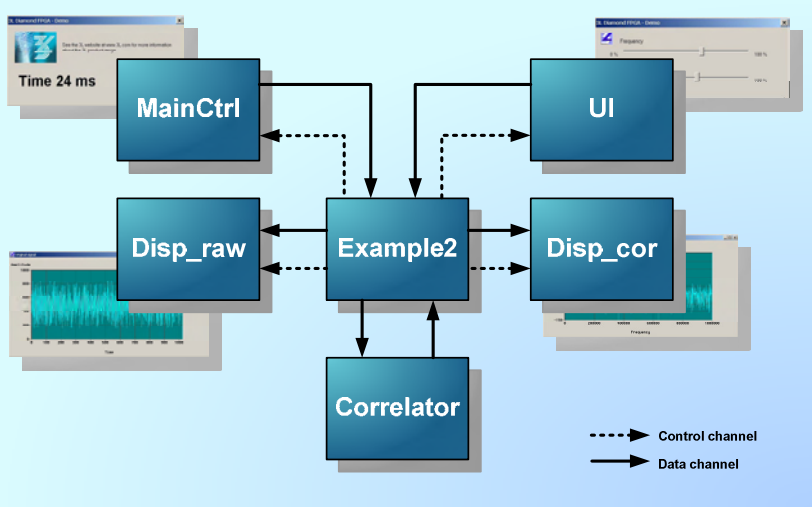
Once the system has been completely specified, the engineer can experiment with where the tasks will run on the actual hardware. Multiple tasks to run on the same processor and the software takes care of loading and other maintenance issues, synchronizing and communicating between processing elements and tasks.



In this manner engineers can easily change system design and implementation by combining tasks in different ways, or placing them on different processors. It is even possible to choose between equivalent implementations for the same task. For example, when a task is placed on a DSP, tools such as Diamond will automatically select the DSP version of that task. When that same task is later placed on an FPGA, then the FPGA version of that task is used. The output of the tool is a single application file that contains all the tasks for the complete system. Because everything is self-contained, things 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 studio streams.

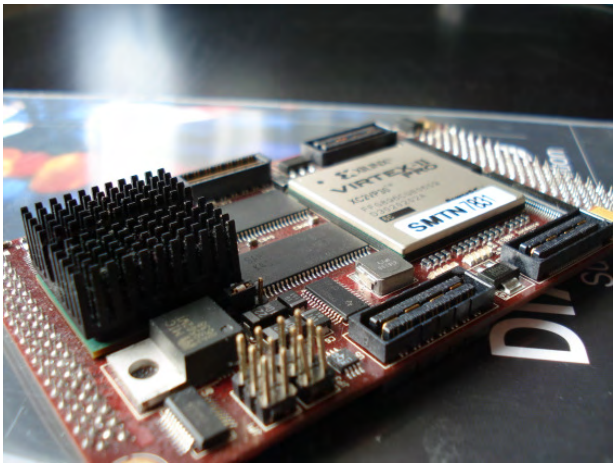
A 6 Task Example System

The following example shows a system made up of six tasks. Some of these tasks represent windows on the host system, which display graphics or prompt the user for input.



The Example2 task creates a sine wave with some random noise added to it. The sine wave is then sent to a correlator task, which calculates the correlation of the sine wave. The result of the correlation is provided back to Example2, which then sends the result to a display task. The original signal is also sent to a display task so that we can see the original signal as well as the result of the correlation. The software makes it possible to easily re-use tasks, for 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. Because the toolset forms an abstraction layer to the hardware, it is trivial to change the actual hardware with something different. The SMT395 module has a C6416T @ 1GHz and a Xilinx Virtex II Pro XC2VP30-6. However the software can also be run on Virtex 4 or 5 versions of similar boards.

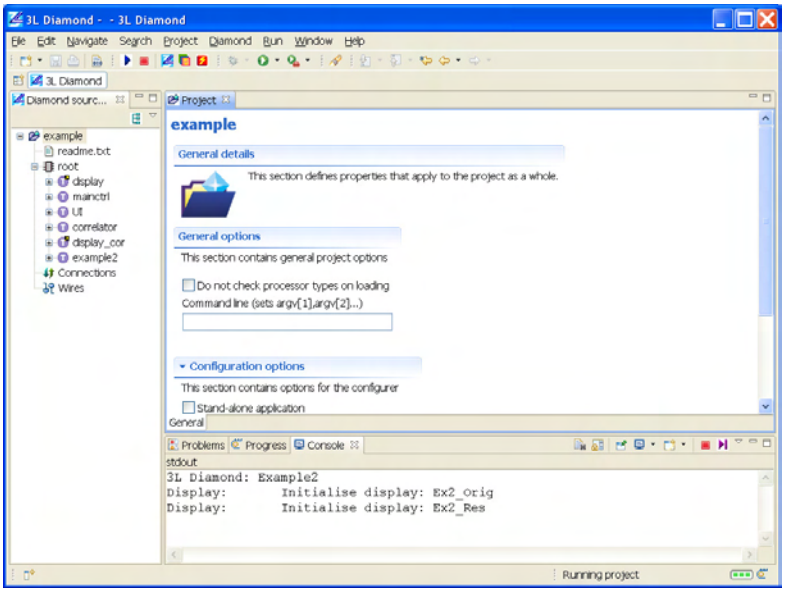


SMT395 FPGA/ DSP Development Platform

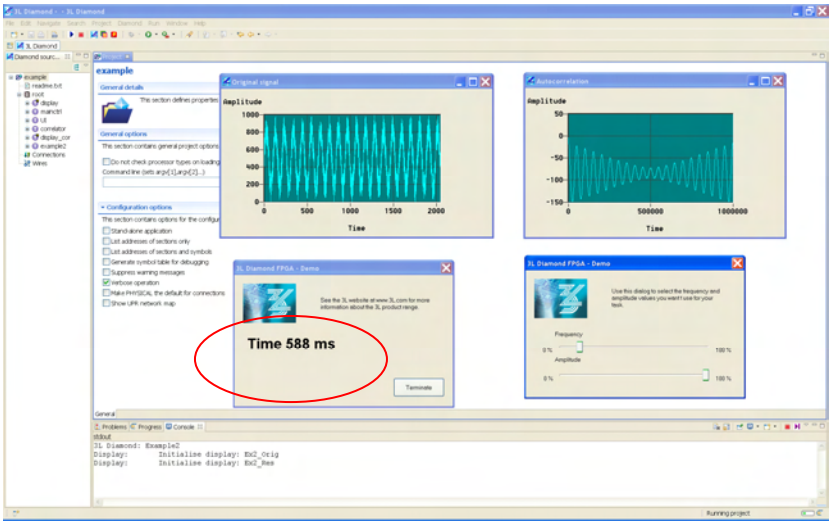
Accelerating Prototyping

During development it is often simpler to develop on DSPs while validating the complete system, even if some parts of the system might not yet be running at full speed. We start by implementing the correlation example by running all the tasks on the DSP. For now we will not place any tasks on the FPGA.

To do this, we use the 3L Diamond IDE. The IDE allows us to specify the tasks and processors in the system. We can also specify how the tasks are connected, and where they should run. The screen-shot shows the six tasks that make up our system all placed on processor root that is the DSP.



When running the application, we get this result.

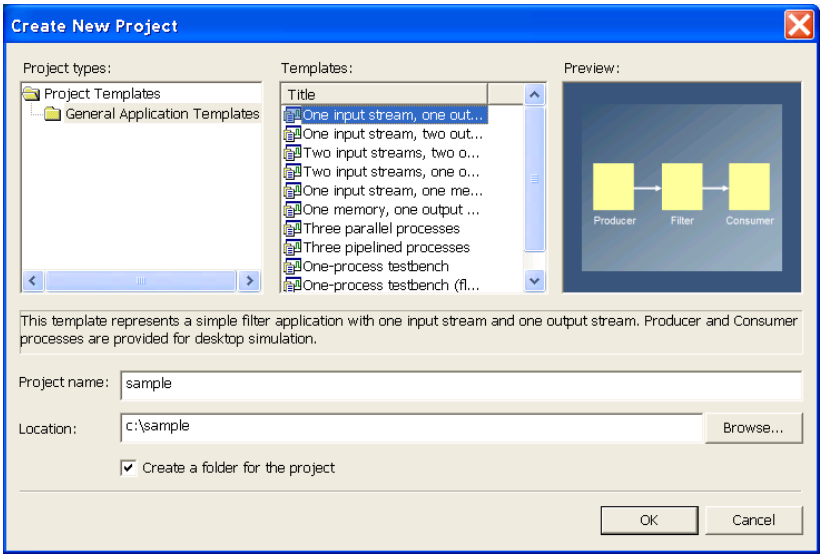


We can see that the correlation takes 588 ms to compute.

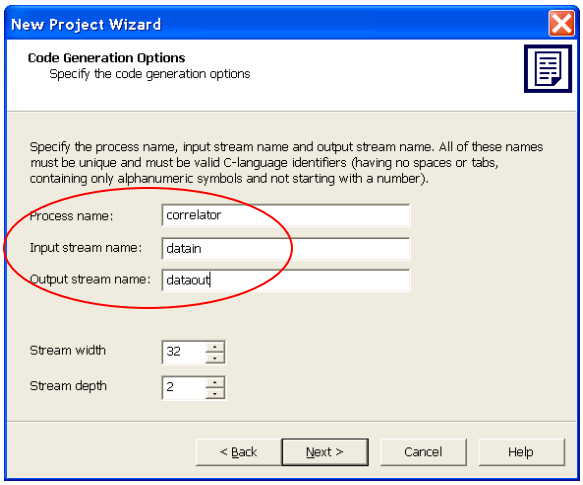
Some processing, such as correlation, is well adapted to the highly parallel environment of an FPGA. Although well adapted for FPGAs, most people find it simpler to develop and test these algorithms in C. To avoid recoding into an HDL, Impulse-C is used to allow development and testing in C which can then be compiled into DSP, GPP or FPGA. Impulse-C tools create an FPGA version of the correlation task that is moved task onto the FPGA of the Sundance SMT395.

Correlation on the FPGA

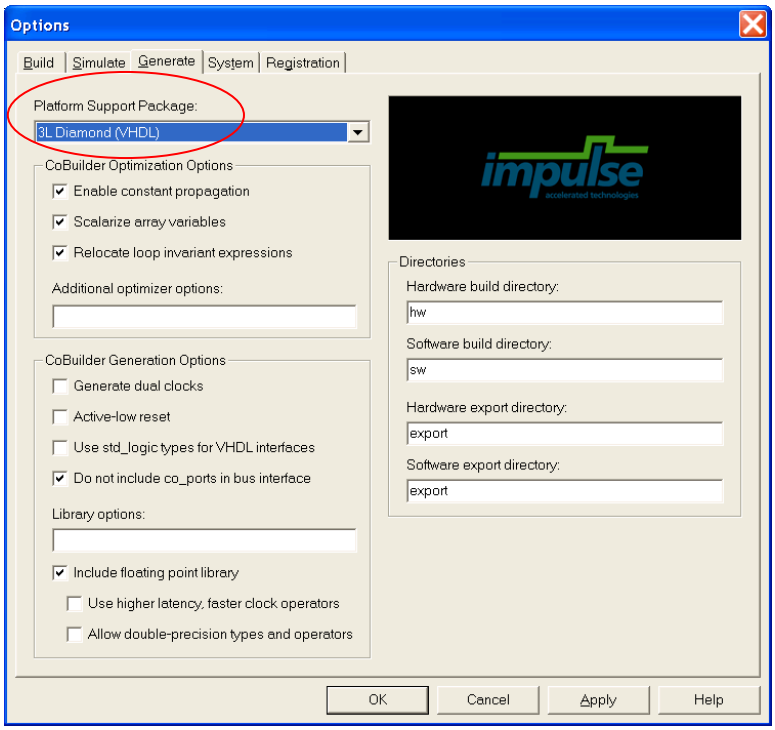
This starts by creating an Impulse-C project, and we specify a template of one input and one output. This input and output are essentially the ports of the correlator task.



Next, the process is named “Correlator”.

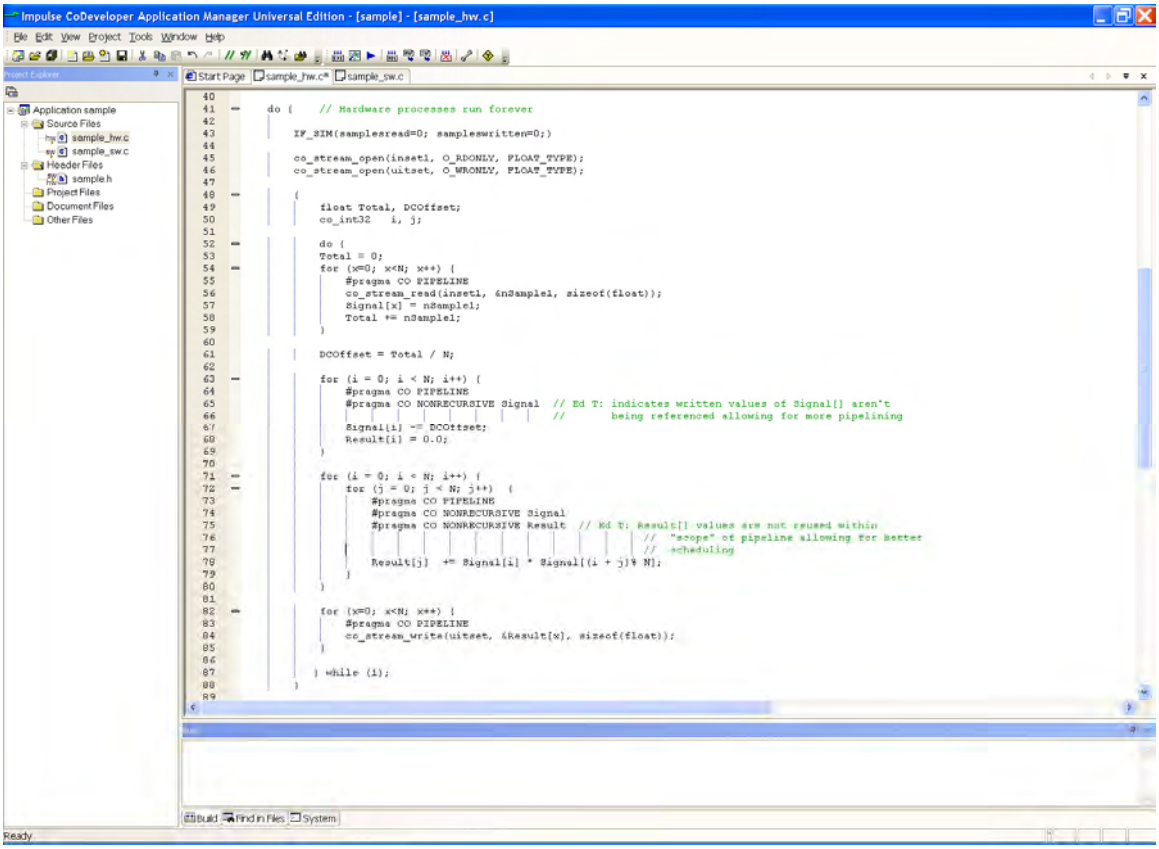


Since the correlator uses floating point, the “include floating point library” option is selected.



We want to be able to use the correlation task in the 3L Diamond environment, so we select “3L Diamond (VHDL)” as the platform support package.

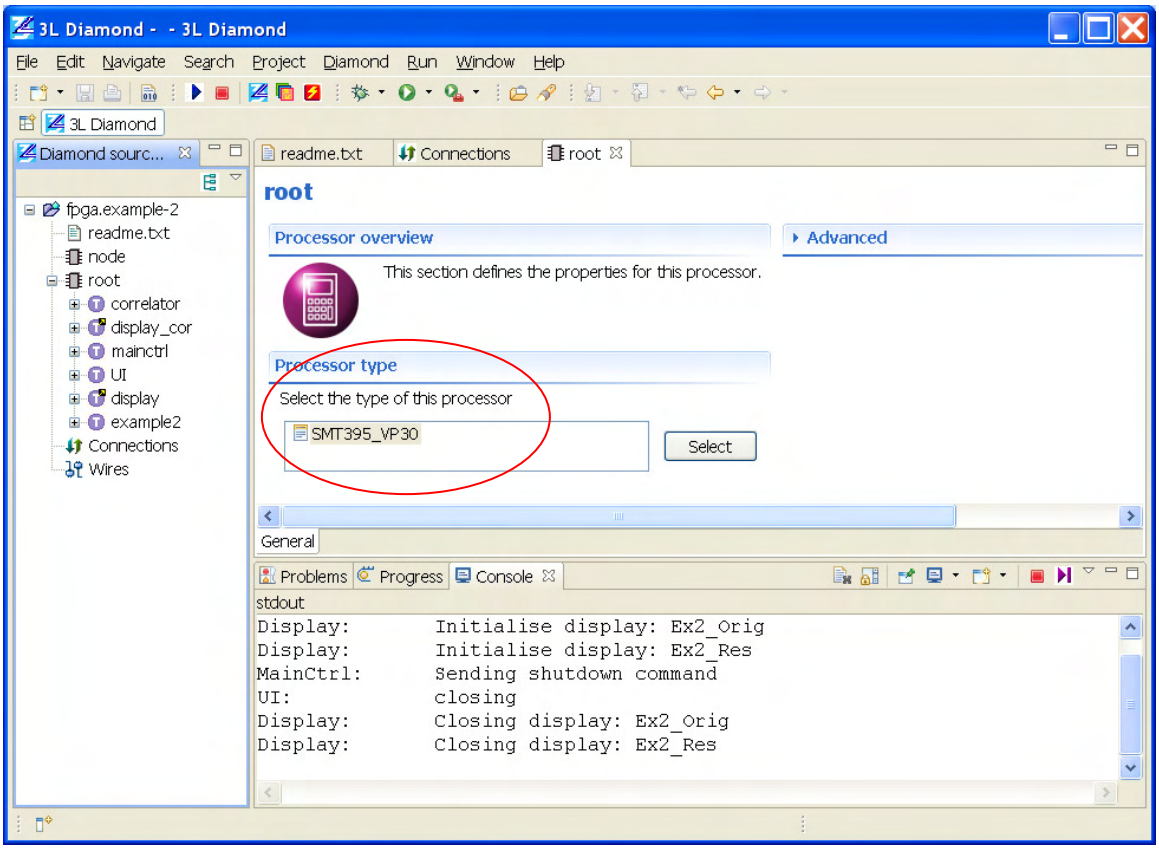
Next the existing C code for the correlation task is imported into the Impulse-C environment.



After several rapid iterations to experiment with the algorithm, a Diamond task is generated by selecting “Generate HDL” from the Impulse-C tool. The resulting task can be placed on any Diamond compatible hardware platform, including the Virtex 4 based SMT8121 and the SMT351T featuring Xilinx Virtex 5 FPGAs.

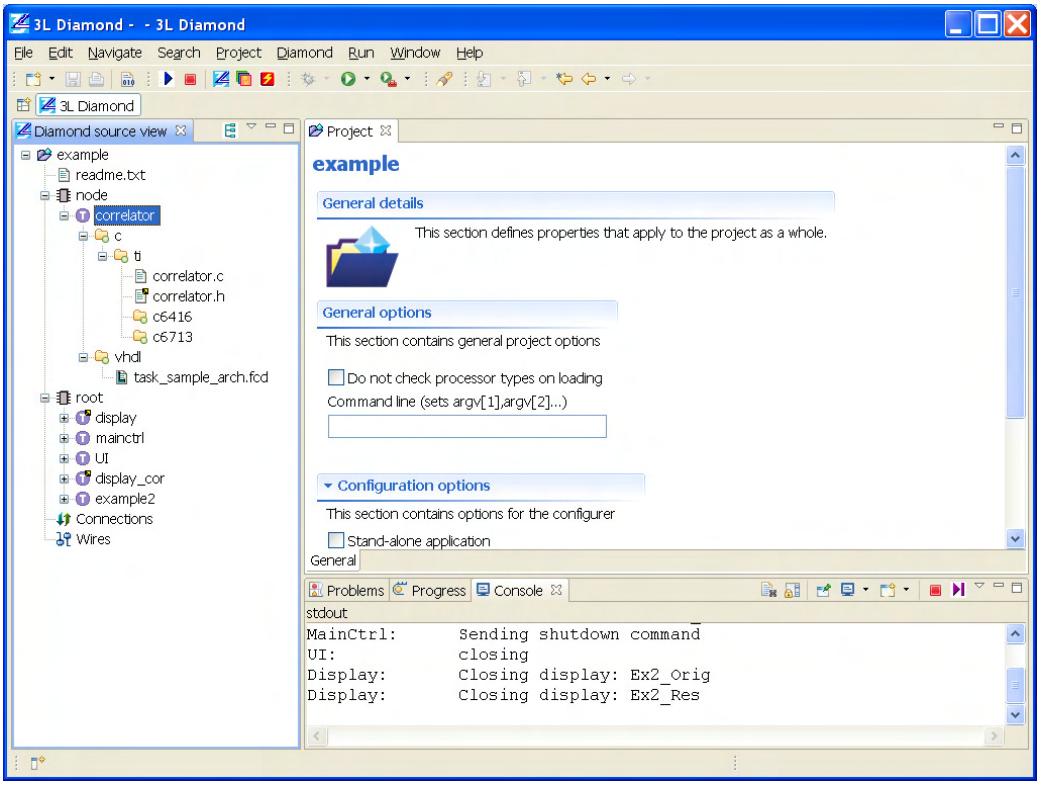
Placing the Correlator task on an FPGA

Returning to the Diamond IDE the FPGA is declared as a processor “node”. The FPGA needs to be specified now since the plan is to place a task on it.



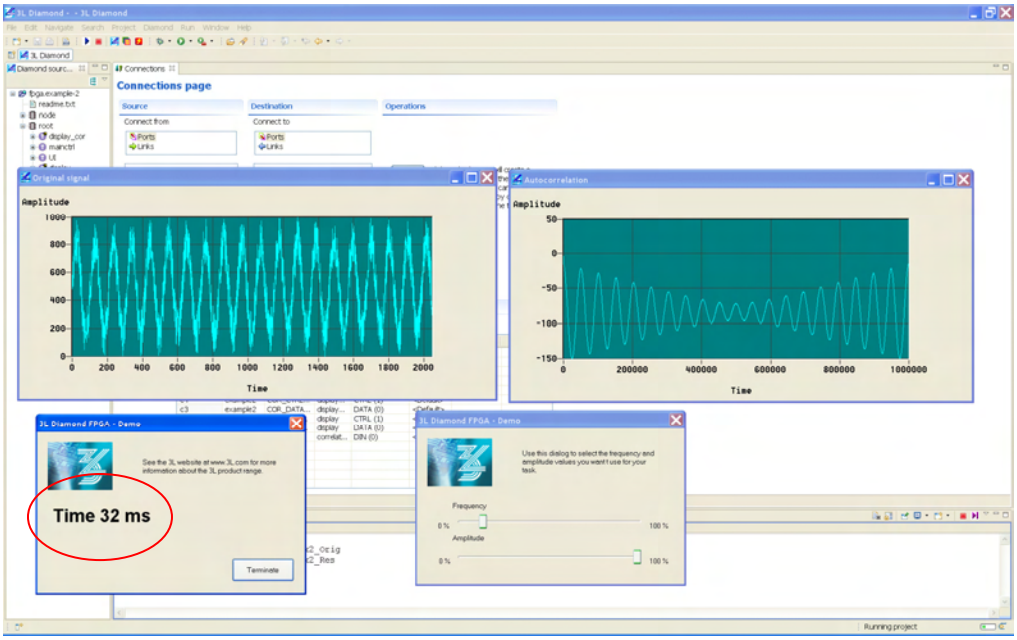
Next the correlator task is dragged and dropped onto the FPGA. Nothing else needs to be changed since the system design remains unchanged. The tasks are still interconnected in exactly the same way they were before. The designer is merely changing where tasks run.

To select the Impulse-C generated version of the correlator task, we point the Diamond tool to the Impulse-C project’s output.



There are now two essentially equivalent implementations of the same task. Diamond automatically selects the implementation appropriate for the technology you place the task on.

When this application is built and, the following result emerges:



By moving the correlation task onto the FPGA, the correlation now only takes only 32 ms, a speed improvement of almost 20 times.

Summary

In this article we showed 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; hardware that yields dramatic performance improvement.