From C Using FPGAs

Paul Goossens

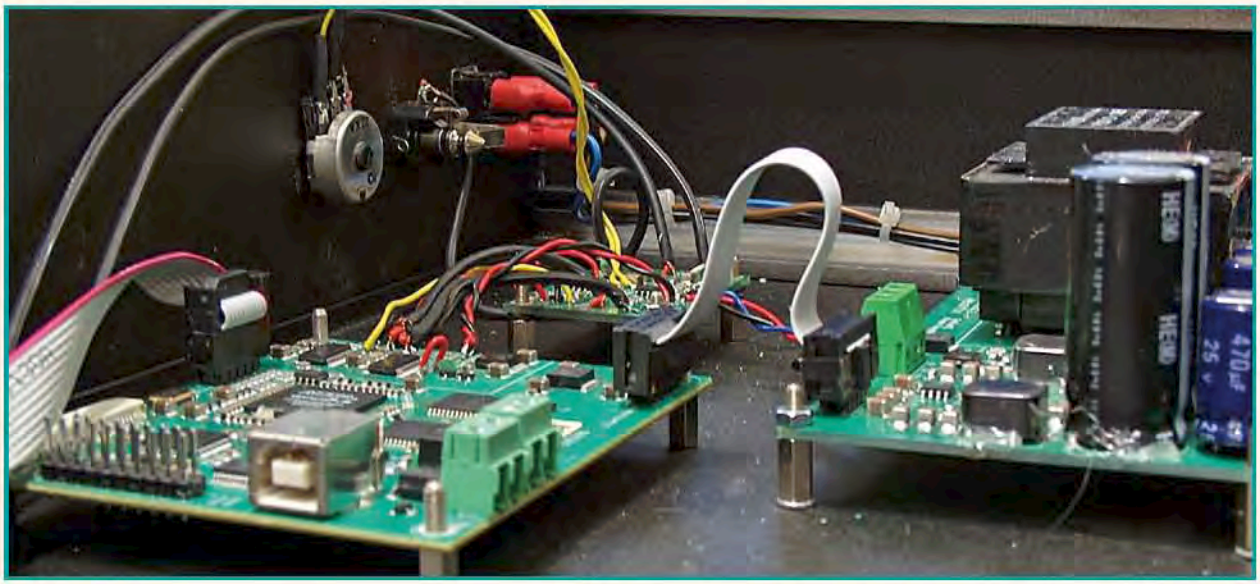
FPGA technology has evolved rapidly in just a short time, and FPGAs are now suitable for a broad range of applications. Nevertheless, most new equipment does not take advantage of the capabilities of these ICs. The main reason for this is that there are still relatively few FPGA experts. The latest developments in the EDA area can largely eliminate this obstacle.

Most of the equipment and devices in our everyday surroundings have a microcontroller inside. Even the simplest products often incorporate some sort of microcontroller. This was not always the case. In the early days of microcontrollers, their use was limited to a select group of engineers, and most electronic equipment was implemented using fully analogue electronics.

The mysterious FPGA

Not too long ago, microprocessors were only used in computers. The engineers who worked with these ICs were called 'experts' and occupied a special field separate from everyday electronics. The associated firmware was primarily programmed in assembly language. Designers with a more analogue bent (which meant the vast majority of designers) regarded microprocessors and microcontrollers as mysterious components.

The same situation exists today with FPGAs. Although most designers have heard of FPGAs and are in fact interested in these promising ICs, they continue to use their tried-and-true microcontrollers in their designs. In many cases a microcontroller is also the most logical choice, but in more and more cases an FPGA could be a good replacement for a microcontroller.



to Hardware

and compilers

Progress

Compared with microcontrollers, FPGAs have the reputation of being expensive and power-hungry. FPGA manufacturers have responded to this by reducing the energy consumption of their ICs and developing inexpensive FPGAs. A lot has already been achieved in this area, and the specifications will only get better in the future.

If you look at the websites of various FPGA manufacturers, you will notice that every manufacturer offers low-cost and low-power families of FPGAs. As a result, the choice between a microcontroller and an FPGA is gradually becoming more difficult if your only criteria are low power and low cost.

Another major objection to using FPGAs is that there are still relatively few designers who are familiar with developing FPGA-based designs.

The most commonly used programming language for programming microcontrollers is C, which operates entirely according to the sequential principle. By contrast, the contents of FPGAs are primarily designed using VHDL or Verilog. These two languages do not belong in the sequential-logic camp. Switching from C to one of these languages requires a completely different way of thinking.

This forms a reasonably large challenge to the designers. If you want to be reasonably proficient in a language such as VHDL, you will certainly have to invest time in acquiring knowledge and experience. This means that your first few FPGA-based designs will take a lot more time than an equivalent design based on a microcontroller. For many companies, this is reason enough to use a microcontroller as a standard solution.

Two worlds

Even if the decision is made to use an FPGA, in many cases it is also accompanied by a microcontroller. This can be an external microcontroller or a microcontroller implemented in the FPGA.

This makes it possible to combine the best of both worlds. With this approach, a large part of the design can still be developed in the traditional manner. The FPGA is used for a specific part of the design, with the rest of the job being handled in firmware in the familiar manner.

Ready-made blocks of 'intellectual property' (IP) are joined together to form a working basis. Sometimes a bit of new design (developed using VHDL, Verilog, etc.) is added to the mix. In many EDA environments, everything can be put

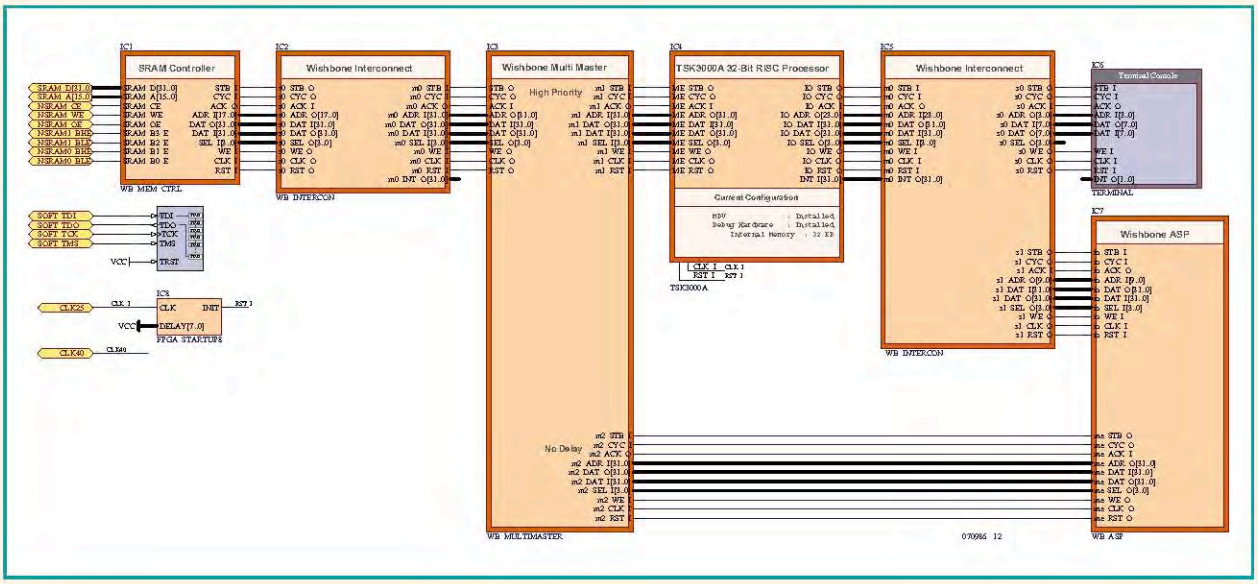


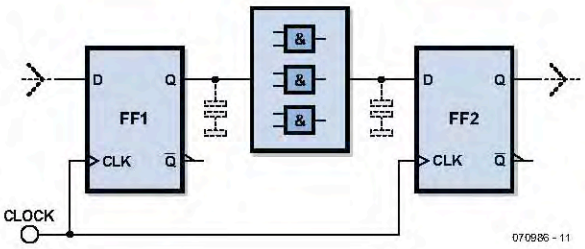
Figure 1.
Internal structure of the FPGA design.

Static versus dynamic current consumption

The current consumed by an IC can be divided into two parts. One part of the current will always flow as long as a supply voltage is present. This current consists of various leakage currents as well as currents necessary for things such as maintaining the contents of static memory. This current is called the 'static current'.

The other part of the current is called the 'dynamic current'. This current results from changing the levels of the various signals inside the IC.

As you know, a digital IC (and thus an FPGA) consists of a collection of simple digital circuits. Each of these circuits has an output. In digital circuitry, each output can have one of two states, which are called '1' and '0' or 'high' and 'low'. The outputs are connected via conductors in the IC to one or more inputs of other logical circuits in the IC.



We know from physics that every conductor has a certain capacitance relative to other conductors in its vicinity. These unintentional parasitic capacitances are often sources of problems for electronics engineers and hobbyists, and they can cause problems in ICs as well. Each time an output changes state, the voltage level of the output signal changes.

As the output has a certain capacitance relative to ground and other conductors, a current must flow in order to charge or discharge this capacitance and thus change the voltage level.

If an output rises from 0 to +5 V, the output circuit must supply a current to charge the parasitic capacitance until it has reached the level of 5 V. If this output switches from +5 V to 0 V, the charge on the capacitance must be discharged to ground in order to return the voltage level to 0 V.

The amount of charge necessary charge the capacitance depends on two factors. The first factor is the difference between the '0' and '1' voltage levels. If the supply voltage is reduced, the amount of charge that is necessary is also reduced. The second factor is the size of the parasitic capacitance. This capacitance can be reduced by modifying the internal structure of the IC and selecting the right materials. Here again, reducing the capacitance also reduces the amount of charge necessary for changing the voltage level.

Synchronous logic

In a synchronous design, the IC responds at the speed of a clock signal. The outputs of the flip-flops can change when a clock pulse appears. If there are combinational circuits following the flip-flops, they will then have different inputs. As a result, their output levels may also change.

After a small time delay, all of the outputs of the flip-flops and combinational circuits will have reached their final states. These outputs can only change when the next clock pulse arrives.

From Hz to current

All this means that the dynamic current consumption can be reduced by reducing the clock frequency. This means that there are fewer clock pulses per second, and this translates directly into a reduction in the dynamic current.

The dynamic current can be reduced even further by ensuring that as few outputs as possible change state on each clock pulse. This can be achieved by shutting down parts of the IC that are not doing anything useful at a particular time. With a clever design, a considerable power saving can be realised using this technique.

Glossary

ASP – Application Specific Processor. Digital circuitry designed to perform a specific task.

EDA – Electronic Design Automation. A collective name for programs intended to simplify the design and development of electronic equipment.

FPGA – Field Programmable Gate Array. An integrated circuit composed of a large number of small logic circuits (in many cases several ten-thousands). Using a programmer, they can be linked together to form large, complex digital circuits. It is even possible to build complete microcontrollers, video cards and the like in this way.

IP – Intellectual Property. In the context of FPGAs, this means software code (such as an 'IP core') that can be purchased, usually with a licence. This code has a predefined complex function that can be used conveniently in an FPGA design.

together using a graphic interface without writing a single line of VHDL or Verilog code.
These methods ensure that the designers do not have to invest too much time in the VHDL design.

From C to H

The latest development, which for convenience we can call 'from C to H', is the next step in VHDL-free design of FPGA circuitry.
Several EDA developers have developed methods to design digital circuitry without using VHDL. The most remarkable method involves direct conversion of standard ANSI C source code into a digital design. There were already compilers available that could convert a specific subset of the C language into a digital circuit. The latest compilers can handle the full scope of standard C (ANSI C).
Two suppliers have now developed compilers that can do this: FPGA manufacturer Altera with its C2H compiler, and EDA developer Altium with its CHC compiler. It is expected that other suppliers will quickly launch their own C to H compilers.

FPGAs for the masses

The strength of these compilers is that routines developed in the old, familiar manner can now take advantage of the power of the FPGA.
Microcontrollers are by their nature developed to be as versatile as possible. This enables them to perform a very wide variety of tasks. This is also one of the reasons for the considerable success of microcontrollers, but it is also one of their weaknesses. Microcontrollers can do everything, but they are not specialists.
FPGAs, like microcontrollers, can do everything – but at a different level. You can build you own digital circuit (including a microcontroller if you will) into an FPGA, but you can also turn it into a specialised bit of hardware that excels in a specific task. These bits of hardware can be developed in C by using these 'C to H' compilers. This means that even programmers with no experience in VHDL can take advantage of the power of FPGAs.

In practice

How does this work in practice? In our lab we can use Altium Designer, which from version 6.8 onwards is supplied with a C to H compiler called 'CHC'. Using this software, we used a simple example to see how various things work in practice.
Our example (see **Figure 1**) uses only standard IP components from Altium. Using these components, you can create an embedded system in almost no time. The processor (IC4) is a TSK3000, which is a 32-bit processor. This processor is connected via several blocks to other modules, such as external memory, a terminal emulator, and our star player: the ASP. The result produced by the CHC compiler ends up in this block.
Consult the Altium website for more information about the blocks used in the diagram.

Test

To test the design, we needed some hardware in addition to the software and an FPGA design. A future *Elektor* project was a perfect candidate for this. This hardware includes a Cyclone III FPGA along with 256 kB of external memory.

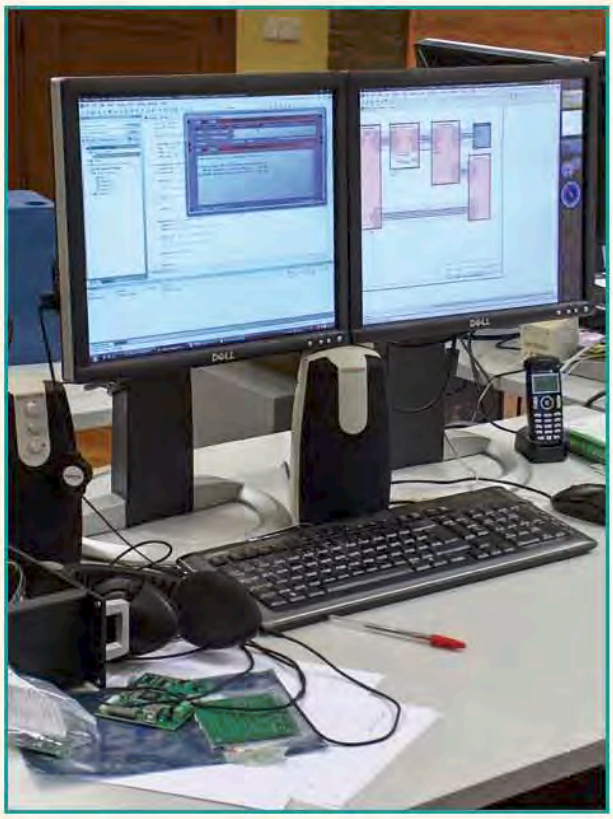
Listing 1: Square root calculation

```
unsigned int isqrt_sw (int number) {
  signed int n  = 1;
  signed int n1 = (((n) + (number)/(n)) >> 1);

  while(((n1 - n) > 1)||((n-n1) > 1)) {
    n  = n1;
    n1 = (((n) + (number)/(n)) >> 1);
  }
  while((n1*n1) > number) {
    n1 -= 1;
  }
  return n1;
}
```

In our test firmware, we have the processor calculate the square root of a number 500,000 times. First we had the ASP (which is the result produced by the CHC compiler) perform this calculation, and we measured the elapsed time. After this, we had the software version of our square root calculation routine do the job 500,000 times and again measured the elapsed time.

The routine we used is shown in **Listing 1**. The first part of the routine uses a clever method to guess the result. The algorithm of this guess is written such that the result is always slightly too large. This value is then used to check whether the result is correct. If it is not, the predicted result is decremented by 1. This procedure is repeated until the correct result is obtained.
The nice thing about this algorithm is that it uses two loops: shift instructions and multiply instructions. This means it contains a nice mix of commonly used operations.



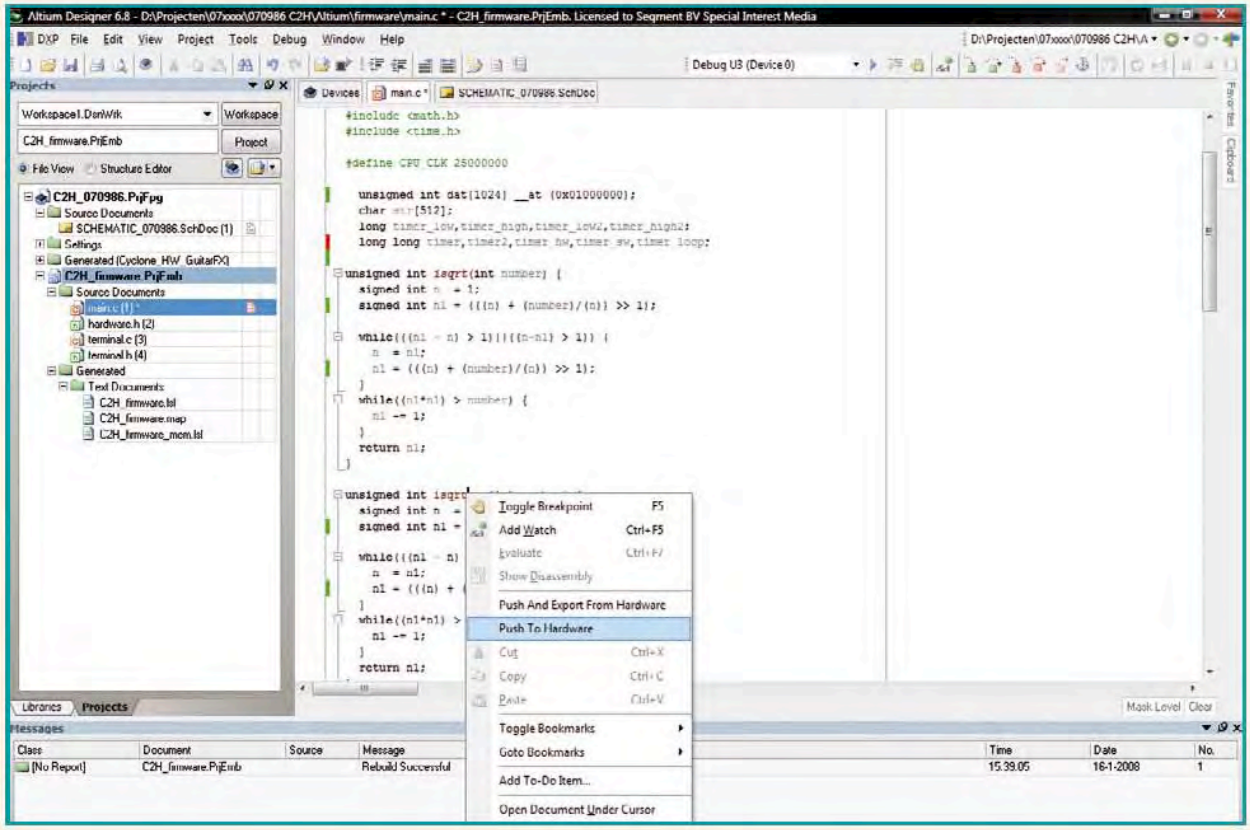


Figure 2.
It could hardly be easier!

Exporting to hardware

We used this routine (called *isqrt_sw*) to measure the time required for performing this calculation. To test the CHC compiler, we made an identical copy of this routine. We named it *isqrt*.

The next step was to use the development environment to cause this routine to be converted to hardware by the CHC compiler. This is a very simple process. First you place the cursor on the name of the routine. If you right-click the mouse, a small menu appears. Select 'Push and export hardware' in this menu (see **Figure 2**). This causes the CHC compiler to implement the selected routine in hardware. In addition, this action causes every call to the firmware of this routine to be replaced by a call to the ASP. All of this takes only one mouse click.

Results

After compilation and programming of the FPGA, the terminal emulator causes the output to be displayed on the PC (**Figure 5**). Here you can see that the hardware version takes slightly more than half a second to perform the square

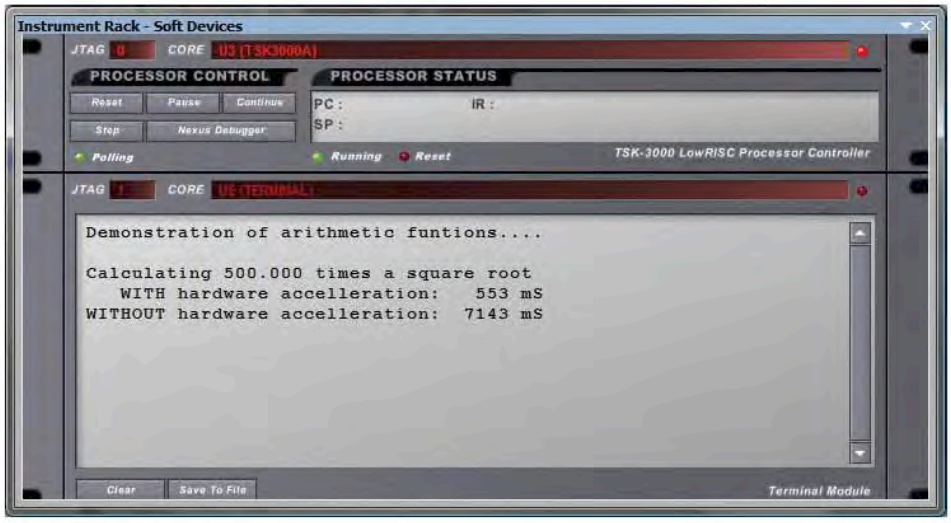


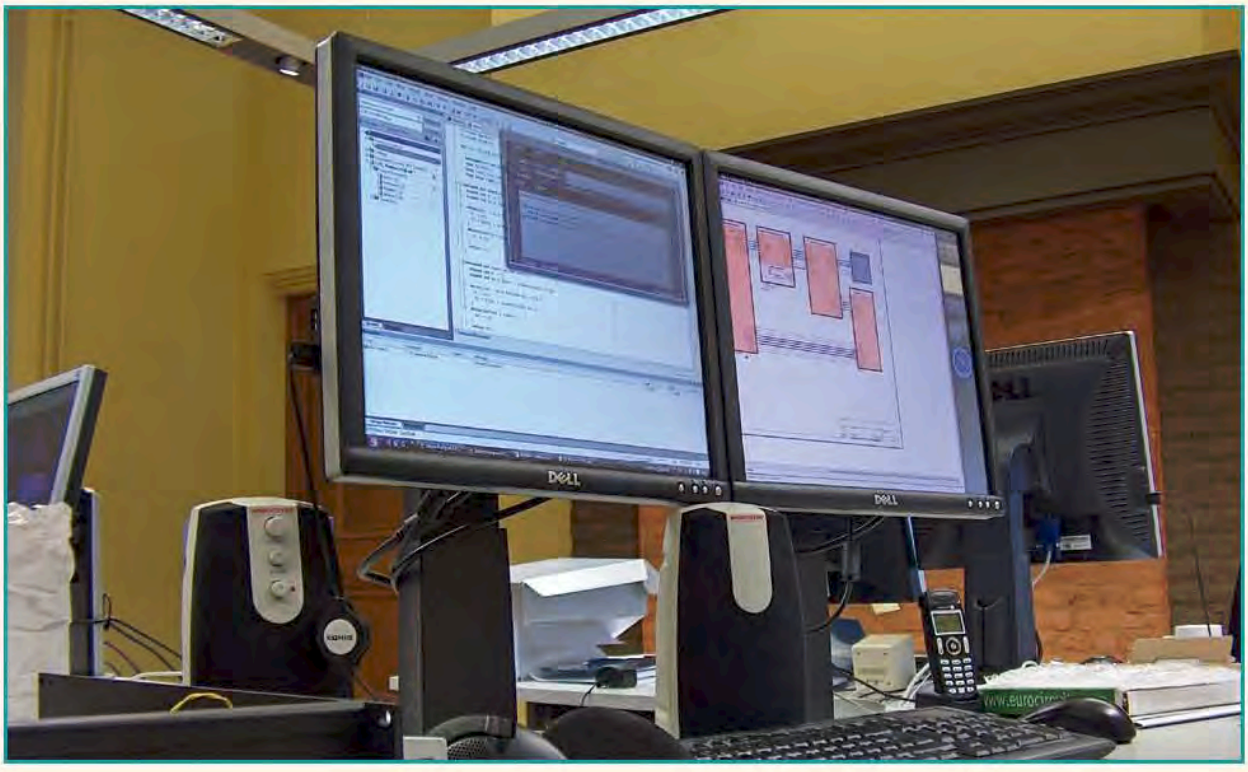
Figure 3.
Test results.

root calculation 500,000 times. The software version takes nearly 13 times as long in this case.
The nicest thing about all this is that with this example we (intentionally) did not write even a single line of VHDL. This technology lets programmers take advantage of the potential of FPGAs without having to leave their familiar C environments.

Applications

The ASP can be regarded as a sort of DIY coprocessor. The ASP can be adapted to individual applications, which means that hardware acceleration can be used in almost any imaginable application.

It is also possible to access the memory from the ASP. This makes it possible to develop applications such as a video accelerator. In this case the speed of the routines for drawing lines, circles and areas could be accelerated by implementing them in an ASP.



It is even possible to have routines in the ASP run in parallel with routines in software. However, this does require modifications to the software. The advantage of this approach is that it further boosts the processing power.
The previously mentioned example of a video accelerator is a suitable candidate for this approach. The processor does not have to wait until the video accelerator has drawn the line, circle or other object, and in between it can spend its time on other tasks.

Low power

A less obvious advantage is that power consumption can be reduced by using an ASP. The clock frequency can be reduced by using an ASP to increase the processing power. The speed gain obtained from using the ASP can be traded off either partially or entirely for a lower clock rate.

The dynamic current consumption of the FPGA decreases in direct proportion to the reduction in the clock rate. The result is thus lower power consumption. This is primarily important in portable equipment and environmentally friendly products.

Final remarks

The new developments in C-to-H compilers make FPGAs accessible to programmers who do not have experience with FPGAs, VHDL and related matters. The reduced development time and the possibility of reusing existing, trusted routines in FPGAs also represent a major step forward in the acceptance of FPGA technology in the engineering world. In combination with the technological progress in FPGA fabrication, this means that FPGAs will become even more competitive alternatives to conventional embedded microcontrollers.

(070986-1)

C-to-H for PCs?

FPGAs are already being used in a few supercomputers in order to assist processor ICs in performing demanding computational tasks.

It is certainly conceivable that in the future, FPGAs could also find a place on the motherboards of standard PCs. For this to be possible, it is necessary to have a standard that specifies how the processor communicates with the FPGA. Even more important is to have a standard technology for converting software routines into hardware versions that are suitable for implementation in FPGAs. The new C to H compilers are a step in this direction. Who knows what speed improvements this technology could offer for future PCs?