

Fast route from system specification to implementation

How can designers speed up their design process? Higher speed is needed to go from specification via design to production while feature-rich implementations increase design complexity. Pressure is increasing – to be the first company to finish the design, start production and grab the major share of the market. Jürgen Pintaske takes a look at the major hurdles.

Only new technologies can improve design productivity – more design with the same size design team. A comparison with the racing car is useful: revs/minute of the engine have a limit, independent of horsepower. The only way to adapt to the revs necessary is to have an adequate gearbox and generate the right power at the right speed. From this analogy new tools have to aim at both the software engineer and the systems designer: if they are to speed the process from specification via simulation and verification to software and hardware integration.

New tools are the only answer, if the limits of existing technology have been reached and other variables are kept constant. The starting point is to accelerate the time to first prototype. The first prototype running software on the prototype hardware is the crucial moment when both designs done by different groups have to work together – and this is the moment when complex errors show up. It occurs late in a standard design cycle, after the extended time required to generate code, partition into hardware and software. Errors found late are costly to correct, even before considering the time lost. Instead a fast route from an executable C simulation to implementation in hardware will find errors quickly, and the cost of correction and re-testing and is kept to a minimum.

The optimum design flow would have an integrated top-down approach with just one tool for the whole design process, both hardware and software. This is impossible to achieve, as so many different targets have to be met in different applications. The next best approach is to implement a fast parallel route to prototype and to use the results to guide the rest of the team in their approved design flow and to avoid mistakes that would be more expensive in design time/cost.

Range of tools

Out of the wide range three options are addressed here which all start basically from C code but are targeted at different areas of the design flow: the Handel-C approach of Celoxica, Bach – developed by Sharp, now being implemented into Mentor Graphics tools and Critical Blue with a co-processor approach starting from compiled C code to speed up selected tasks of the application.

All of these tools use C language as a common starting point, as this is the HLL language most widely used in the embedded industry and most engineers know it. Via different routes they help to generate hardware descriptions that can be loaded into fpgas or be used via VHDL for asicsoc design.

The obvious advantage of this approach is to extend existing engineering resources and capabilities to achieve better tested/corrected code faster. By keeping a coherent design environment in C for as long as possible, iterations can be performed quicker. They are less prone to errors as translations by hand, e.g. from C to VHDL, are either left out completely or come later in the design cycle when more code has been generated already with less errors.

An important aspect of hardware design is gate efficiency. But this is not necessarily important at the beginning of the design cycle, since, if the prototype hardware runs on an fpga, then only two aspects are relevant: does it run fast enough and does it fit into the fpga.

For asic design other options have to be considered. How much will a

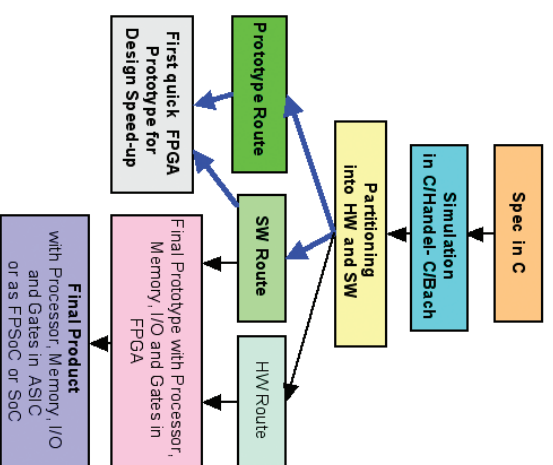


Fig. 1: Simplified system design flow involving HW and SW

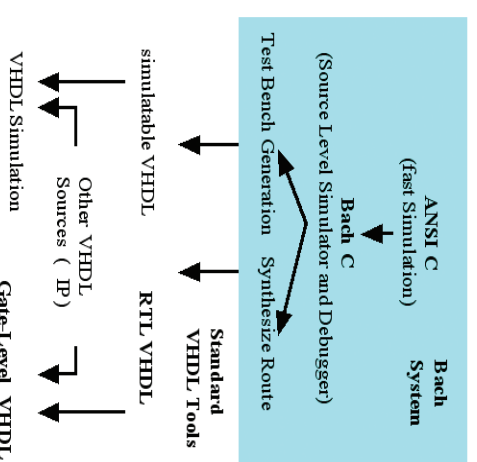


Fig. 2: Bach C design flow targeted at SoC design

gate count reduction achieve or, more importantly, where is the balance between design time and design cost compared to entering production early with a less well-optimised design and optimising as soon as volume actually ramps up.

Design scenario

Many projects start with a specification implemented as a C program for simulation or to generate a C-executable for simulation (figure 1). An easy route from C to gates to give short route to hardware, even for parts of the software, may reduce simulation time by a factor of 1000 or more compared to simulating in software. Speedy location of design errors allows for a quick change and increases the rate of iterations considerably.

Algorithm verification, for example can be a very slow process, especially if multimedia features such as MP3 or image files have to be included. The fastest solution here would be to take the algorithm in C, compile it to an fpga and run it at accelerated speed or even in real-time.

After having defined and simulated all of the functionality, the partitioning process into hardware and software can take place. As much as possible of the C code will have to run on the system processor to save on additional hardware. As processor usage reaches 100 percent execution time, any additional functionality required has to be implemented elsewhere, for example as additional gates in the fpga or asic. As software and hardware designers have co-operated in the debug process, the start of the final prototype will be quicker.

Handel-C

Celoxica's Handel-C is probably the best known way to cover the wide area of converting C language programs to hardware, co-simulation and co-verification. The development kit (DK) has been shipping since 2001. But how does Handel-C work? The background is CSP (Communicating Sequential Processes), an approach used a long time ago with transputers to run software as parallel processes where

communication between different processes was taken care of automatically. As all of the 'inputs', 'outputs' and 'connections' of any piece of code were connected via a special communication protocol, it was not important where the code was running.

Handel-C is based on the same CSP approach and the tool DK converts C functionality to logic blocks in hardware, connected via latches, and optimises the resulting network to generate an EDIF output directly for fpga tools from Altera/Xilinx. Additional Register Transfer Logic (RTL) in VHDL or VERILOG can be generated to feed into an existing design chain for asic or fpga implementation.

The most important fact is that Handel-C is fully synchronous, there is only one master clock. Changes propagate at each rising clock from block to block. Between two clock cycles there is time for the data to settle, until the next clock cycle propagates the data further. The whole design is by definition fully synchronous. All of the code is translated into blocks that execute based on this clock, where latches make sure that everything is running in lock-step. On the basic level this executes like sequential execution (similar to software but faster).

The real speed-up comes with the PAR statement, which defines pieces of code that will switch blocks of gates in parallel. The designer will first define the obvious independent functions to execute in parallel and refine if the required speed cannot be achieved.

If the speed required for the design is not achievable with a single clock domain (one master clock) then multiple clock domains have to be used, where the output of one clock domain will trigger another block of gates. The same approach has to be applied, if peripheral functions are tied to specific clocks.

The general design approach can be compared to software development in C and Assembler: First, design and verify the algorithm/application. If execution time in C is then not fast enough, find critical code and optimise in Assembler (which means introduce more parallel execution of Handel-C). The aim in Handel-C is not necessarily to find the fastest implementation from the beginning, but to achieve a working debugged and tested solution as quickly as possible and then optimise until the required execution speed for the application is achieved with an acceptable amount of gates.

A main advantage of Handel-C is that a top down design can be used from the beginning, but implementation of code in fpga is possible from the beginning as well (figure 1). Fpga boards and integration tools with all the necessary support software are available, so a designer can start with a standard solution without having to build their own boards or interfacing code. As result the design team can get an impression very early in the design cycle about speed of execution in software or hardware.

Partitioning into hardware and software will be a lot easier, as code that took long to simulate will have been ported and debugged in fpga already, so 'hot-spots' are known. It is important to understand, that the Handel-C code will basically stay the same all the time. EDIF for fpga tools of Altera and Xilinx can be generated directly from Handel-C and recoding into a Hardware Description Language like VHDL is not necessary, unlike the normal route from C through VHDL to fpga. The same approach applies to the application code and to any test benches that have to be generated for software and for hardware.

Bach
 Sharp's Bach HLL defines another route to hardware, optimised for different design aspects, developed jointly by Sharp Laboratories of Europe in the UK and Sharp Corporation in Japan. Sharp has used Bach on about 20 asics. These have been successfully put into production with greatly reduced design time.

Soon this technology will be available for general use. Last year Mentor Graphics and Sharp announced that they are working together on building a tool for total system LSI development support from design to verification. Both companies are working together to include the Bach technology in Mentor's ASAP tool.

Bach starts from C like Handel-C, but instead aims to generate highly efficient VHDL for the standard asic and fpga design tool chains. In contrast to Handel-C, which is fully synchronous, Bach uses an 'untimed' (not asynchronous) approach instead.

Since the semantics of Bach C are untimed the Bach synthesiser has more freedom to generate fast VHDL code. Unfortunately this means that the relationship between code and execution time is more difficult to read. The designer cannot tell easily from examining the source code, in which clock cycle any particular operation will happen. The Bach synthesiser ensures though that data cannot get lost due to timing differences.

Unlike VHDL and SystemC the parallelism is not restricted to the top level. A par keyword is used define sub-processes which have to execute concurrently and these sub-processes can contain further par

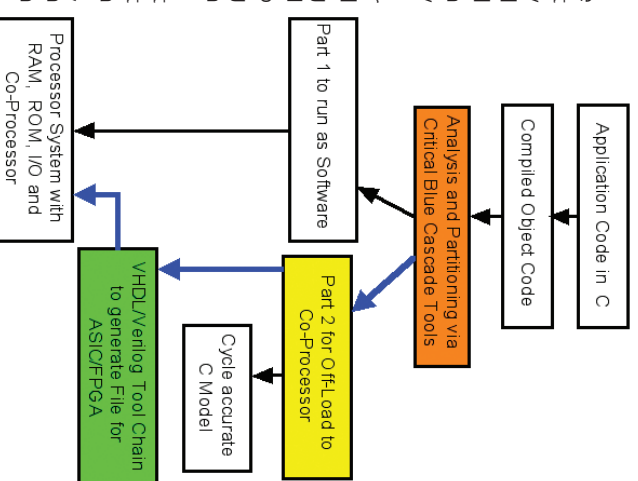


Fig. 3: Cascade design flow for co-processor

statements. At execution time all of the sub-processes are executed concurrently and the par statement finishes when all the sub-processes have terminated.

Communication between different concurrent processes can be defined as synchronous or asynchronous. Data transfer is via send and receive constructs. Both sender and receiver have to be ready for the transfer to happen.

Each achan (asynchronous channel) can be written to and read by various threads, like a global variable. Since it is an asynchronous implementation, the exact time of read or write cannot be predicted during simulation. Both types of channel can be used for communication to external i/o as well.

Bach contains a behavioural synthesiser that compiles the parallel Bach code into several communication modules and automatically generates the i/o interfaces between them. The output is VHDL RTL ready for use as input to standard VHDL tools. Mentor's ASAP incorporating Bach, is currently in beta-test with customers and will be released soon.

Critical Blue

Newcomer Critical Blue comes from a different angle. Here the starting point is compiled application code which is analysed and via automatic generation of parallelism and output of VHDL. The main target is to start from existing object code and to save slices of execution time of the main processor by transferring selected functionality into the hardware of a dedicated co-processor, communicating via the existing processor buses.

If the existing application runs out of resources or the new design cannot execute fast enough on the processor system, the 'overspill' is

offloaded onto an additional asic or fpga. Here the designer does not start from C but from tested application object code. The first implementation of the tool is targeted at ARM code.

The Cascade Tool Suite operates on the compiled output of the standard software tool chain. Object code analysis generates fast feedback on how best to partition the existing application software into processor execution and co-processor support. The automatic generation of a cycle accurate C model of the selected code allows for a rapid analysis of the 'extracted' co-processor performance.

In a next step VHDL RTL code is generated which is processed via the existing VHDL design tool chain for implementation in fpga or asic. Product availability is targeted for the end of 2003.

Software importance

It is a well-known fact that the importance of software has increased compared to hardware and the new tools reflect an even further change. In the past software and hardware departments have been quite separate. Pressure on ever shorter design cycles is forcing the use of new tools that generate code faster. Writing more code faster will be difficult: the only solution is to write code on a higher level of abstraction.

Handel-C starts with C and helps with rapid prototyping in fpga and in partitioning and verification. Bach, implemented in Mentor's ASAP product achieves a faster route to VHDL and asic. Critical Blue's Cascade solution is somewhere in between, e.g. if a processor system needs a turbo charge via a co-processor, but the rest of the system stays unchanged. All of these tools involve the software designer much more in hardware implementation.

The main hurdle will be the design department. Changing an existing design flow is always a risk and in the past a new tool has been accepted only when there is no choice. In such a case of a difficult system design, software and hardware, all have to work together. Future will tell how quickly designers will accept this offer in a more general way. CE