# リアルタイムシステムのためのハードウェアコンパイレーション技術

イアン・ページ　　マット・ニューマン
Celoxica Limited

日本連絡先：　日本セロックシカ株式会社
〒240-0005　横浜市保土ヶ谷区神戸町 134　横浜ビジネスパークウエストタワー11 階

**キーワード**:　ヘンデル C 言語、セロックシカ、FPGA/PLD、リコンフィギャブル、
ハードウェアコンパイレーション、パラレルプログラミング

**概要**：今日リアルタイムシステムには設計者が考慮しなければならないプロセッシングコンポーネント（CPU, DSP, FPGA/PLD,および ASIC）の選択肢が多くある。システムが複雑になればなるほど、プロジェクトの成功は、開発ツールやランタイム環境に依存する。
本稿では、Handel-C 言語と DK1による、生産性向上、かつ、複雑なアルゴリズムの FPGA/PLD へのダイレクトな実装、実行を可能にする新しいアプローチを紹介する。これを、組込みシステム内のビデオ暗号化アルゴリズムのハードウェア加速例を用いて説明する。従来の手法と比べて設計時間が大幅に短縮し、決められた時間でより多くのアーキテクチャ構想が可能になるため、デザインもより改善されたものになった。

# Hardware Compilation Techniques for Real-Time Systems

Ian Page
Chief Science Officer, Celoxica Ltd.
Visiting Professor,
Imperial College of Science and Technology,
University of London

Mat Newman
SVP Technology, Celoxica Ltd.

Keyword : Handel-C, Celoxica, FPGA/PLD, Reconfigurable, Hardware Compilation, Parallel Programming

Abstract:
Many choices of processing components face the designer of today’s real-time systems, including CPU, DSP, FPGA, and ASIC. With such systems increasing in complexity it is often the development tools and runtime environments that make or break a project.
This paper presents Handel-C and DK1 which provide a new approach to rapid design that allows application specialists (software or hardware professionals) to increase their productivity and implement and execute algorithms directly on FPGA silicon. As design is so much faster than conventional methodologies, it is routinely possible to investigate many more architectural possibilities within a fixed design timeframe and so produce improved designs. This is illustrated using an evaluation project where hardware acceleration of a video encryption processing bottleneck was implemented using these tools and the subsequent throughput improvement assessed.

# Introduction

A system designer is faced with a bewildering array of system components when architecting real time systems. The processing requirements alone may require one or more technologies, such as CPU, DSP, FPGA, and ASIC. Selecting the appropriate balance of components for speed, cost, flexibility and time to market can be a daunting task, and implementation is no less so. With such systems increasing in complexity it is often the development tools and runtime environments that can make or break a design project.

Handel-C and DK1 provide a new approach to rapid design that allows application specialists (irrespective of whether they are software or hardware professionals) to increase their productivity and to implement and execute algorithms directly on FPGA silicon. Using this design methodology is so much faster than conventional design flows, that it is routinely possible to investigate many more architectural possibilities within a fixed design timeframe. The natural outcome of this is the creation of designs that are significantly improved in terms of size, speed or functionality. Indeed it is not uncommon in our experience to find designs that are improved in all three areas simultaneously.

FPGAs already have an established role in prototyping new systems. As their density and performance increase and prices decrease, they are becoming increasingly viable also for production systems. Taken together with our Handel-C based design approach, FPGAs closely coupled to conventional processors provide a cost-effective, flexible and powerful system architecture.

Our basic approach is to use a software programming language for defining system functionality. In our view, it is crucial that the language is used in the same manner that a software designer uses a conventional programming language. This is in contrast to using a conventional language to describe hardware architecture. The latter approach may (or may not) help hardware engineers to design hardware architectures, but it does nothing at all to help software engineers to realize their systems in hardware, and it does almost nothing to help software and hardware engineers to work together on defining functionality.

By using Handel-C, a software language closely conforming to the ‘C’ language, we enable software and hardware engineers genuinely to work together on creating functionality, whether it is delivered in hardware or software form. This can massively increase productivity, particularly given the skills shortage in hardware design and the costs and timescales usually associated with hardware design.

There are two essential innovations in the design of the Handel-C language which distinguish it from C. Firstly is the introduction of the ‘par’ construct which allows the designer to specify parts of the code to be executed in true (rather than pseudo) parallelism. In abstract terms, this allows the designer to control the extent of the implementation in space. The other innovation allows the designer to control the temporal characteristics of the implementation. This is done by a single Handel-C language definition such that each and every assignment statement takes exactly one clock cycle to execute, and that no other language structure takes any time at all. With these two features, the designer has control over the time and space characteristics of the implementation. Controlling both time and space simultaneously is precisely what a hardware designer does using a hardware description language. In the case of Handel-C programming it is done at a much higher level of abstraction; that is how productivity is improved.

We couple our Handel-C design approach with implementations using a combination of conventional processor and FPGA technologies. This means that total systems (that is both the hardware and the software) can be compiled onto standard platforms within minutes. Far from this being a mechanism only for fast prototyping,

our customers are already seeing that we support a new implementation paradigm for them, namely ‘the prototype is the product’.

In this paper, we illustrate our approach by describing the results of a collaborative project with Wind River Systems to produce a reference design (hardware and software infrastructure) to explore the concept of closely coupled reconfigurable logic and microprocessors. The chosen application was the real-time encryption, simulated transmission, decryption and finally display of a video stream.

## Hardware Platform

The hardware architecture chosen was based around the PowerPC processor and Xilinx Virtex FPGAs. In order to get the project running quickly, development was started using two COTS (Commercial Off The Shelf) parts. These were a PPMC750 Single Board Computer from Wind River, and an RC1000-PP from Celoxica. The PPMC750 single board computer contained a PowerPC 750 processor, and was plugged into a PCI backplane to connect to an Ethernet card and RC1000-PP FPGA card. The RC1000-PP is a PCI based Xilinx Virtex FPGA card with 8mb of local memory.

Whilst initial development for the software was being undertaken on the above configuration, a new reference platform based on the PowerPC 405GP processor was developed. This “405GP” board has PCI connectors (in both slot and PMC forms) into which FPGA based cards are plugged. This card also had the advantage of a custom connector that allows an FPGA daughter card to be plugged directly onto the processor’s peripheral bus, thus allowing higher bandwidths to be achieved. Figure 1 shows a block diagram of the 405GP board. The software infrastructure provides a layer of software and hardware functionality for every new platform introduced. This ensures that porting an application from one hardware platform to another becomes little more than a re-compilation with the appropriate platform library.

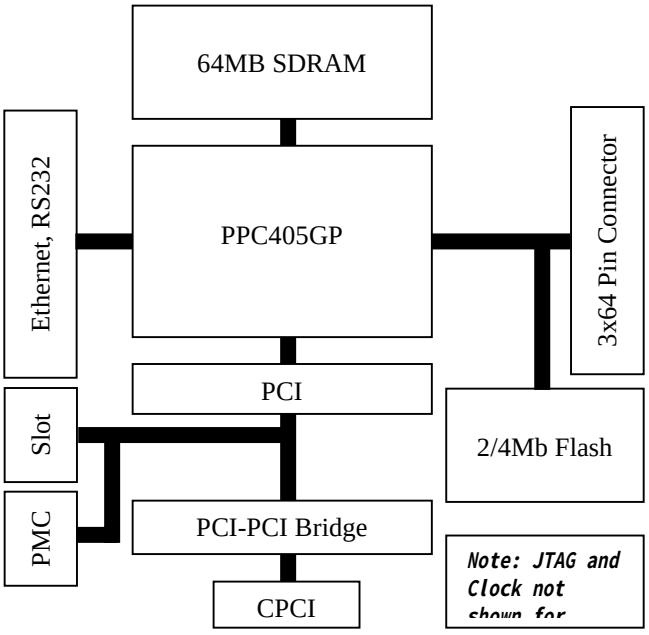


**Figure 1**

A variety of FPGA daughter cards were used with the system, including the ADM-XRC card from Alpha Data Systems, and the Proteus card from Wind River. For these cards a simple DAC interface was also provided so that we could use the FPGA to drive a video monitor. The finished system was completed with a flat panel LCD display so that it could run stand-alone demonstrations.
In this implementation the performance is measured by monitoring the amount of video data being passed through the FPGA and displaying the results on an LCD screen attached to the FPGA board. The display part of the application within the FPGA contains a sync generator for providing signals to the VGA adapter. In order to meet the timing requirements of the 640 * 480 VGA display and to simplify system design, the speed of the FPGA clock was set to exactly 25.407231 MHz.

## Software Development

We report on two major aspects of the software development for this project. The first is the tooling used to develop applications for the system, and the second is the run time environment that controls the system during operation.

### Development Tools

We chose to use the VxWorks real time operating system on the 405GP processor. We also used various hardware bring-up tools from Wind River such as VisionClick and VisionDesktop. These tools allow for close control of the boot cycle for the board in the time before control is passed to an operating system. We also used the PAVE API from Xilinx to program the FPGA with configuration files.
Application content for the FPGAs was developed using the Handel-C language and the DK1 development suite. This allows C based code to be synthesized into a netlist that is suitable for FPGA implementation using vendor Place and Route Tools.

## Run Time Environment

One of the main issues we wanted to address was the ease with which developers with little or no hardware knowledge should be able to develop applications for the hybrid CPU+FPGA system. We did not wish the developer to be burdened with having to program at a low level of detail. Typically the FPGA is connected to the CPU in a memory-mapped fashion, and we did not want the users to have to develop their own communication protocols and data marshalling routines. We therefore developed a co-processing API that would define how C code running under VxWorks on the CPU could access ‘hardware functions’ running in the FPGA.

### Data Streaming Manager

The API is implemented as a Data Streaming Manager (DSM). There are two parts of the DSM; one residing on the processor (software) side (the S-DSM) and one residing in FPGA hardware (H-DSM). In our initial implementation, the hardware functions are server functions that wait for data input before processing begins. As far as the developer of hardware functions is concerned, the interface is relatively straightforward; a standard RPC (Remote Procedure Call) in fact. The hardware function reads parameters from an input port and then writes results data to an output port. The complexity of receiving commands over the PCI bus, routing parameters to the correct hardware function, and then routing the responses back to the appropriate calling software thread is handled entirely by the H-DSM.
On the software side there are two main aspects of the co-processing API, namely the configuration and then the actual usage of the custom hardware function. Properties relating to the S-DSM are held in a DSM_T structure, and contain information on the base address of the FPGA in the processor address space, the number of functions available, status information etc. From the base address, the rest of this information is retrieved by

the S-DSM from the FPGA by reading a memory mapped register. This lets the S-DSM know which functions are contained in the current FPGA configuration.

## Video Application Example

Our main application example in this project was based around streaming, compressed, encrypted video. We used the FLI (AutoDesk animator pro file format for simple graphic animation) format for video compression, and implemented an FLI player in hardware using Handel-C and DK1. A cartoon animation file is loaded into the memory on the PPMC750 card and used as reference data for encryption. A triple DES algorithm was coded up in C and runs on the Power PC. The same code was also hand-converted into Handel-C and run in hardware. The comparison between the software and hardware implementations of this triple DES function was a goal of the project. A 64-bit key is used for the encryption and this same key allows correct decryption. Implementing three single DES algorithms in sequence to produce Triple DES encryption further increases the integrity of the DES standard. Naturally, three separate 64-bit keys are used for encryption and decryption. The triple DES algorithm is inherently sequential in software but can be heavily pipelined in hardware for increased performance.

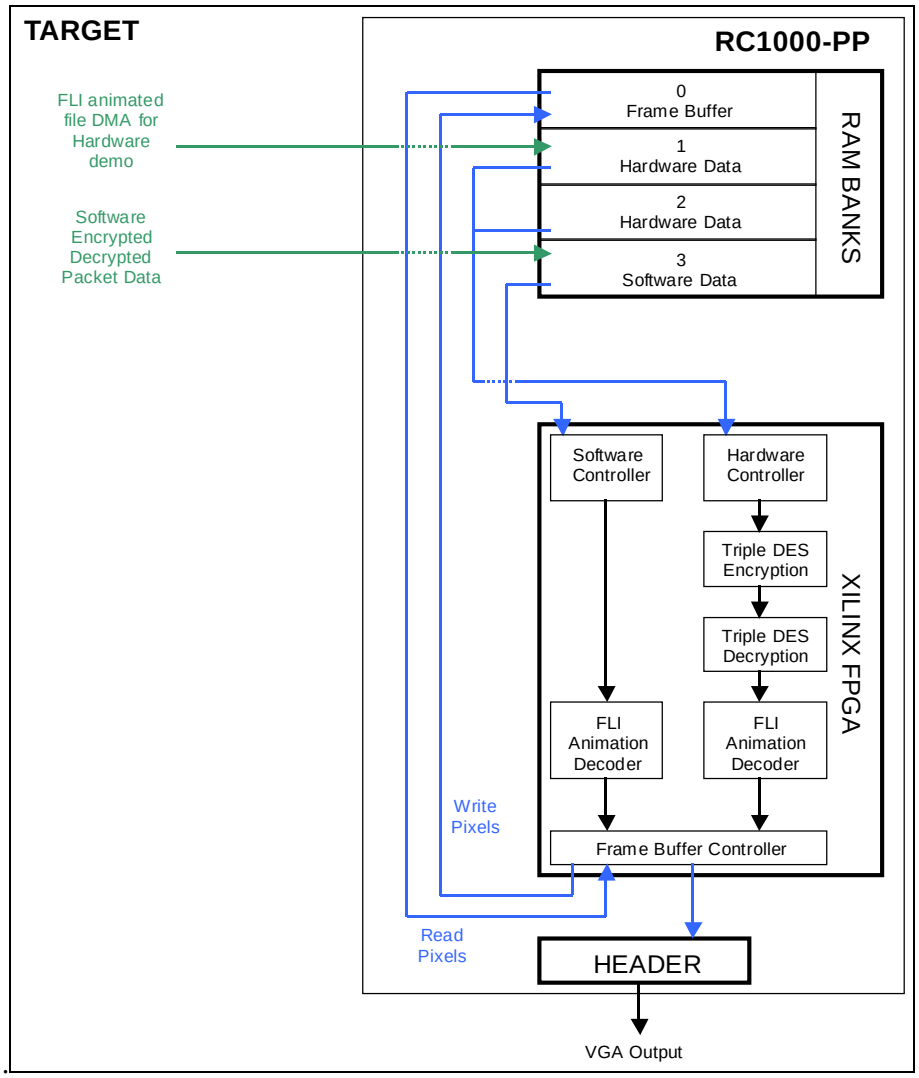


**Figure 2**

In operation, the FLI cartoon specification file is decoded in FPGA hardware by a Handel-C program, with each frame being encrypted, decrypted and displayed on a VGA monitor. Both software and hardware cycles are triggered at the same time with the hardware cartoon being programmed to cycle continuously until the software one finishes. The processed data from the microprocessor is fed to the FPGA that has also been programmed to generate VGA signals. The output from both the hardware and software implementations is then merged to form a composite image on the monitor. A block diagram showing data flow around the FPGA subsystem is given in figure 2.

## Performance Comparisons

In order to measure the comparative performance of the hardware and software implementations of the triple DES application we constructed a test harness that streamed data into either a software or hardware encryption algorithm.

The theoretical performance of the system can easily be calculated since the triple DES implementation produces a 64-bit word every 19 clock cycles. This fact is known by inspection of the Handel-C program as it is simply a matter of counting assignment statements. Since the FPGA clock runs at a known rate of 25.4MHz in order to satisfy the VGA display requirements, this immediately gives a data throughput of 85.6Mbps. The system's actual performance was profiled using WindView, an application that lets the user set trigger events at different points in the code and then receive accurate timing information of when each trigger event occurred. The performance figures are given in the table below.

| | **Encryption** | | **Decryption** | |
|---|---|---|---|---|
| | Software | FPGA Co-processing | Software | FPGA Co-processing |
| Elapsed time for 1MB of data | 5,558.8 ms | 424.8 ms | 5562.9 ms | 424.7 ms |
| Cryptography rate | 1.51 Mbps | 19.7 Mbps | 1.51 Mbps | 19.8 Mbps |

It can be seen from these figures that, although the hardware implementation gives an impressive 13 times speed improvement, it is still only running at 25% of the theoretical maximum rate, showing that in the total system performance the triple DES encryption is nowhere near being the bottleneck. Note that this speedup is also achieved using a 15 times slower clock. This is not an unusual result to have when implementing a software algorithm directly in hardware; that is the power of true parallelism. What really is unusual is that hardware implementation can be achieved quickly, reliably and effectively by software engineers. That is the true power of Handel-C

## Conclusions

We initially faced several issues relating to designing for this mixed system. Some of these were addressed by creating new tools or APIs, whilst others are still largely a manual process performed by the designer. Specifically:

- *How to partition the application between software and hardware?* This is currently a purely manual process. In Celoxica it is the subject of further research and future tooling may address certain partitioning issues. Profiling of the final system showed that the full speed-up benefits of placing core routines in hardware may be compromised by other system bottlenecks, such as RAM access or bus speeds. In our

examples we ported larger blocks of functionality (such as a complete triple DES core, and a complete FLI decoder) to the FPGA. Other architectures could be explored, such as porting fragments of these applications to the FPGA.

- *How to port applications written in C to an FPGA?* Normally creating functionality in an FPGA would require hardware design skills and knowledge of a Hardware Description Language such as VHDL or Verilog. These are generally outside the skill set of embedded software developers. By using Handel-C and DK1 we are able to create FPGA functionality from a software programming language. The source code for the software and hardware implementations are then substantially similar, and software engineers may tackle both. Development times are similar to those using regular C. As an example, the FLI player took 2 man-weeks to implement in hardware, as did the triple DES functionality. The integration of these two blocks to produce the cartoon demonstration took only half a day.

- *How to configure the FPGA with suitable functionality?* Low-level hardware bring-up tools support configuration of the FPGA from bit files produced by the DK1/Xilinx P&R flow. The Xilinx PAVE API supports run-time configuration of FPGAs from within a C application running under VxWorks.

- *How to handle communications between the CPU and the FPGA?* In order to simplify the co-processing aspect of the system we developed an API and associated implementation in both VxWorks and Handel-C. This provides a flexible mechanism for offloading functionality from a processor into an FPGA with minimal overhead on the part of the designer. This system is also bus neutral, in that moving from say a PCI connection between processor and FPGA to a direct memory mapped connection would be transparent at the level of application source code.

- *How to measure where system bottlenecks are?* Using the WindView tools enabled close monitoring of the performance of the CPU. Matching up calls to the FPGA showed the overheads associated with offloading functionality into hardware. Further timing analysis in this case showed that this was due to the latencies of setting up and executing the PCI bus data transfer. By using DMA transfer of larger blocks the overall throughput could have been increased, although this would require additional functionality to be incorporated into the FPGA. A better alternative is to connect the FPGA directly to the CPU bus in a memory mapped fashion. This is the next step in our development plans.

- *How to simulate and debug the combined system?* Linking processor simulators with the Handel-C simulator allows for the co-simulation of both hardware and software portions of the system. This is an area for further work to provide closer integration of a wide array of simulators, and also to support both functional simulation and cycle-accurate simulation.

- *How to provide portability of designs across different platforms?* Having a defined API for communications between the CPU and the FPGA greatly assists in the portability of applications. At the application level it can completely abstract away from underlying architectural choices, such as PCI bus vs. peripheral bus. Using high-level languages (such as C, C++ and Handel-C) provides for a rapid means of porting code from one processing platform (CPU or FPGA) to another.

Combining reconfigurable logic with microprocessors can enable significant speedups in system performance. With the right development tools and run time environment the process of developing applications that take advantage of the hybrid system is greatly simplified. However, care is needed in choosing which functionality to place in the FPGA if the speed gains from hardware implementation are not to be swamped by the overhead of transferring data to and from the FPGA and CPU.

## Acknowledgements

Much of the implementation work detailed in this paper was undertaken by two software engineers at Celoxica, namely Stephen Chappell and Hammad Hamid. The paper is a reworking of a previous one written by Mat Newman, who was at the time SVP Technology for Celoxica Limited. The other author has trimmed the earlier paper, and added material on the design philosophy and approach embodied in Handel-C and is responsible alone for any errors or omissions.

## Author Biography

Mat Newman was until recently SVP Technology for Celoxica Limited. He holds a BA in Mathematics and Philosophy, and two Masters degrees in Computer Science. He has worked in the software development and consulting industries for over ten years before joining Celoxica in early 2000.

Ian Page had early industrial experience of parallel and real-time systems, followed by a twenty five year academic research career in digital electronic and silicon design, computing and electronic design automation. He initiated work on ‘Handel’ languages for hardware compilation in 1990 as an academic at Oxford University. He founded the Hardware Compilation Research Group at Oxford and was the technology founder of Celoxica. He is currently an independent consultant and Visiting Professor at Imperial College, London.