



Application Defined Processors

By Dan Poznanovic
Created 2004-12-01 00:00

By rebuilding a system's logic on the fly, this project can make one FPGA do the work of tens or hundreds of ordinary processors.

Application defined processors are based on the concept of reconfigurable computing (RC). RC is a computing technology that blurs the line between software and hardware and provides the basis for the next big steps forward in delivering high performance with reduced power and space requirements. RC is implemented using hardware devices that can be reconfigured. Processors in an RC system are created as hardware that is optimized for the application that executes in it.

This article explains RC, examines SRC systems that implement RC and shows the performance advantage RC provides over traditional microprocessors. We also explore the programming model for RC and discuss the potential RC provides for supporting Open Hardware.

What Is Reconfigurable Computing and Why Do I Care?

RC is a form of computing based on hardware that can be created dynamically for each application that will run in it. RC hardware is comprised of chips whose logic is defined dynamically rather than at the time the chips are fabricated. RC has been around for many years and implemented in a number of different hardware components, such as field programmable gate arrays (FPGAs), field programmable object arrays (FPOAs) and complex programmable logic devices (CPLDs). What is important to application developers is that today's reconfigurable chips have a clock rate and capacity that make it practical to do large-scale computing with RC hardware.

The most familiar chip type used to implement RC is the FPGA. An FPGA is a chip composed of SRAM memory cells used to define a configuration for the chip. FPGAs contain logic gates, flip-flops, RAMs, arithmetic cores, clocks and configurable wires to provide interconnection. FPGAs can be configured to implement any arbitrary logic function and, therefore, can be used to create custom processors that can be optimized to an application.

So, a collection of FPGAs could be configured to be a MIPS, SPARC, PowerPC or Xeon processor, or a processor of your own design. In fact, the processor need not even be an instruction processor. It could be a direct execution logic (DEL) processor that contains only computational logic requiring no instructions to define the algorithm.

DEL processors hold great potential for high performance. A DEL processor can be created with exactly the resources required to perform a specific algorithm. Traditional instruction processors have fixed resources, adders, multipliers, registers and cache memory and require significant chip real estate and processing power to implement overhead operations, such as instruction decode and sequencing and

cache management.

DEL processors are reconfigurable computers created for each application in contrast to a fixed architecture microprocessor where one size fits all. A DEL processor delivers the most efficient circuitry for any particular application in terms of the precision of the functional units and parallelism that can be found in the algorithm. Being reconfigurable, a unique DEL processor can be created for each application in a fraction of a second.

But why do you care that a DEL processor can be created dynamically for an application, and that it uses its chips more effectively than a microprocessor? The answer is simple: performance and power efficiency. A DEL RC processor can be created with all of the parallelism that exists within an algorithm without the overhead present in a microprocessor. For the remainder of this article, RC processors are assumed to be implemented using FPGAs in order to be more specific in the discussion.

How Is that High Performance Achieved?

Performance in RC processors comes from parallel execution of logic. RC processors are completely parallel. In fact, the task of constructing the logic for a given algorithm is to coordinate the parallel execution such that intermediate results are created, communicated and retained at the proper instants in time.

A DEL processor is constructed as a network of functional units connected with data paths and control signals. Each computational element in the network becomes active with each clock pulse. Figure 1 shows a fragment of logic for computing an expression and contrasts the utilization of the chip versus a von Neumann instruction processor, like the Intel Pentium 4 microprocessor.

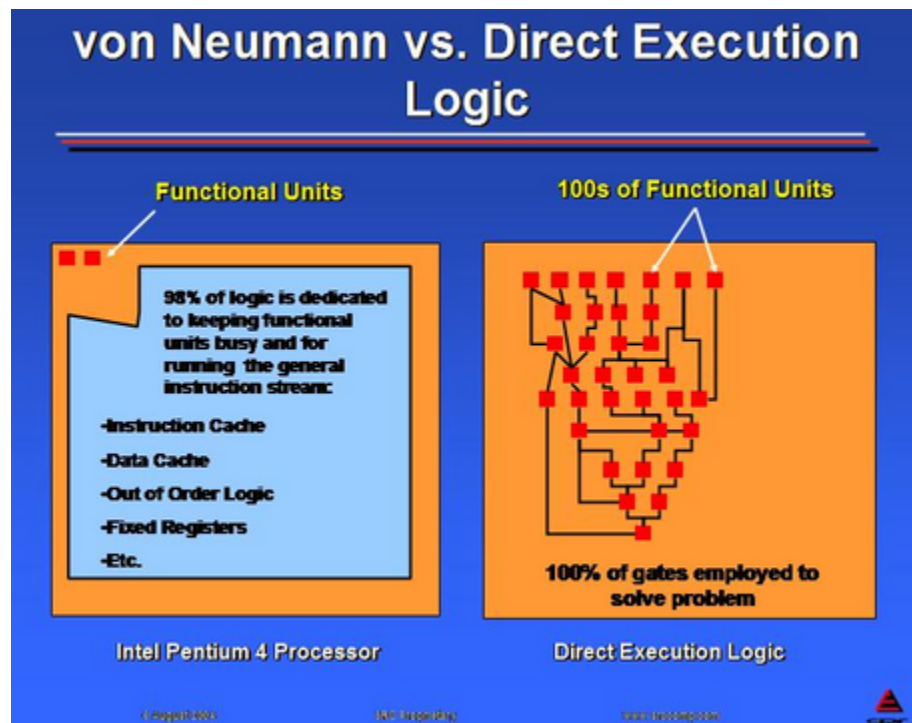
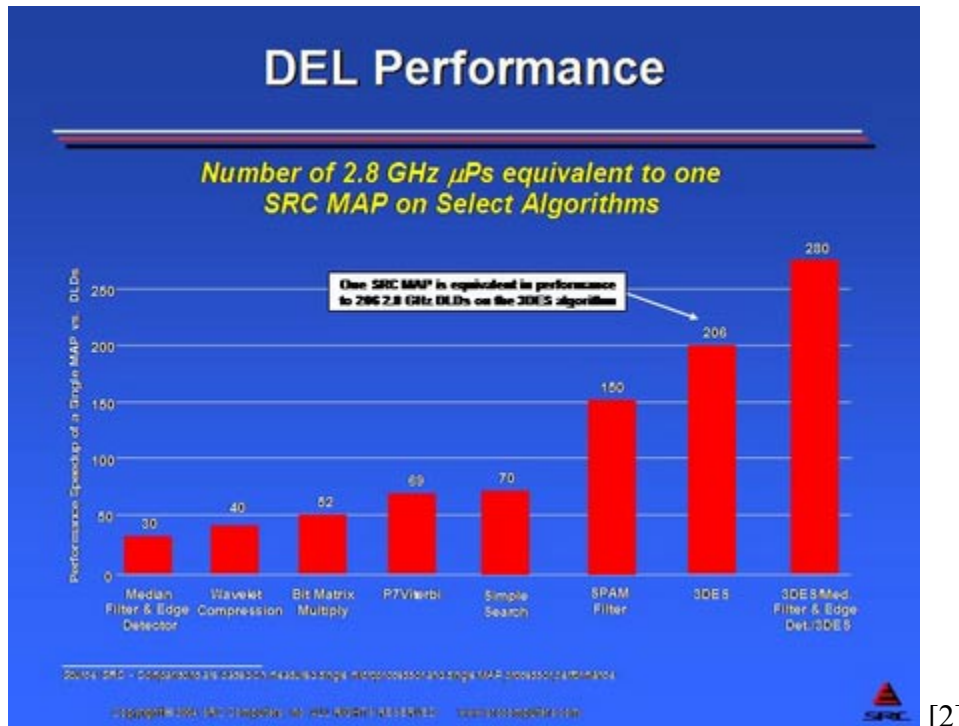


Figure 1. Direct execution logic can put all logic gates to work on the real problem.

Even though a microprocessor can operate at a clock frequency of 3GHz and the FPGA chips operate in the 100-300MHz frequency range, the parallelism and internal bandwidth on a DEL processor can outperform the microprocessor by orders of magnitude better delivered performance. Figure 2 presents some benchmark comparisons between SRC's DEL processor, MAP, and a typical von Neumann instruction processor, the Intel Xeon 2.8GHz microprocessor. Parallel execution of exactly the required number of functional units, high internal bandwidth, elimination of instruction processing overhead and load/store elimination all contribute to overcoming the 30× difference in clock frequency between the MAP and the Intel microprocessor.



[2]

Figure 2. Number of 2.8GHz microprocessors required for the same performance as a MAP direct execution logic processor.

But Can a DEL Processor Run Linux?

DEL-based processors could run Linux, but do they need to? Code segments within the Linux kernel certainly might benefit in performance from running on a DEL processor, and applications within the Linux distributions also could achieve higher performance. However, the role of an operating system, and the kernel in particular, is to manage the hardware such that applications achieve their required performance levels. In other words, the OS is supposed to stay out of the way and let applications consume the hardware.

Applications do a lot more than intense computation. They interact with users, read and write files, display results and communicate with the world through Internet connections. Thus, applications require both computational resources and the services of an operating system. Heavy computation with high parallelism benefits from DEL processors. Although serial code could run as DEL, it is best serviced in a traditional microprocessor.

The best combination of hardware for running most applications is a mix of microprocessor and DEL processors. This combination allows applications to achieve orders of magnitude performance gains

while still running in a standard Linux environment with all of the OS services and familiar support tools. The portion of an application that is predominantly sequential or that requires OS services can run in a traditional microprocessor portion of a system, while applications and even portions of the OS that benefit from the DEL parallelism run on a closely coupled DEL processor.

SRC Computers, Inc.'s RC System

SRC has created systems that are composed of DEL processors and microprocessors. SRC systems run Linux as the OS, provide a programming environment called Carte for creating applications composed of both microprocessor instructions and DEL, and support microprocessor and DEL processor hardware in a single system.

The DEL Processor-MAP

The patented MAP processor is SRC's high-performance DEL processor. MAP uses reconfigurable components to accomplish control and user-defined compute, data prefetch and data access functions. This compute capability is teamed with very high on- and off-board interconnect bandwidth. MAP's multiple banks of dual-ported On-Board Memory provide 11.2GB/sec of local memory bandwidth. MAP is equipped with separate input and output ports with each port sustaining a data payload bandwidth of 1.4GB/sec. Each MAP also has two general-purpose I/O (GPIO) ports, sustaining an additional data payload of 4.8GB/sec for direct MAP-to-MAP connections or data source input. Figure 3 presents the block diagram of the MAP processor.

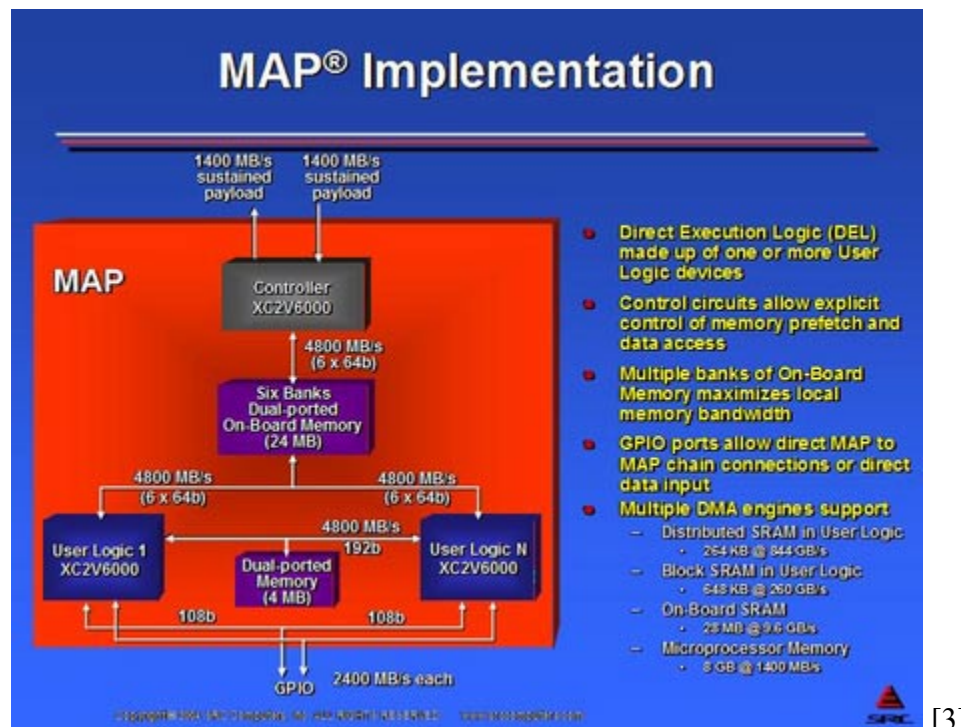


Figure 3. Block Diagram of MAP

Microprocessor with SNAP

The Dense Logic Devices (DLDs) used in these products are the dual-processor Intel IA-32 line of microprocessors. These third-party commodity boards are then equipped with the SRC-developed SNAP interface. SNAP allows commodity microprocessor boards to connect to, and share memory with, MAPs and Common Memory nodes that make up the rest of the SRC system.

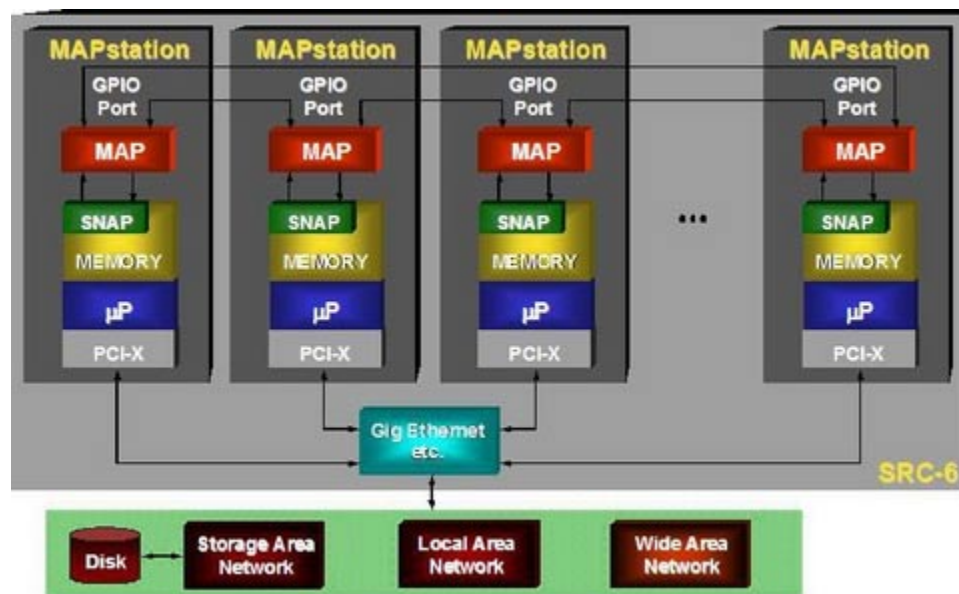
The SNAP interface is designed to plug directly in to the microprocessors' memory subsystem, instead of its I/O subsystem, allowing SRC systems to sustain significantly higher interconnect bandwidths. SNAP uses separate input and output ports with each port currently sustaining a data payload bandwidth of 1.4GB/sec.

The intelligent DMA controller on SNAP is capable of performing complex DMA prefetch and data access functions, such as data packing, strided access and scatter/gather, to maximize the efficient use of the system interconnect bandwidth. Interconnect efficiencies more than ten times greater than a cache-based microprocessor using the same interconnect are common for these operations.

SNAP either can connect directly to a single MAP or to SRC's Hi-Bar switch for system-wide access to multiple MAPs, microprocessors or Common Memory.

SRC-6 System-Level Architectural Implementation

System-level configurations implement either a cluster of MAPstations or a crossbar switch-based topology. Cluster-based systems, as shown in Figure 4, utilize the microprocessor and DEL processor previously discussed in a direct connected configuration. Although this topology does have a microprocessor-DEL processor affinity, it also has the benefit of using standards-based clustering technology to create very large systems.



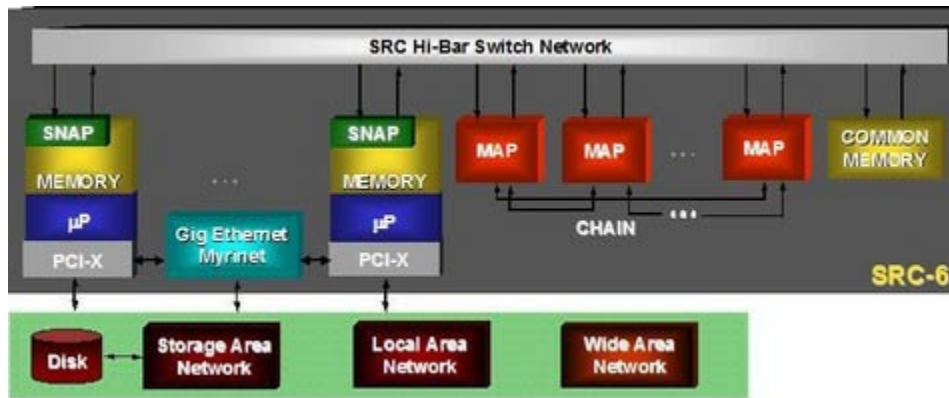
[4]

Figure 4. Block Diagram of Clustered SRC-6 System

When more flexibility is desired, Hi-Bar switch-based systems can be employed. Hi-Bar is SRC's proprietary scalable, high-bandwidth, low-latency switch. Each Hi-Bar supports 64-bit addressing and has 16 input and 16 output ports to connect to 16 nodes. Microprocessors, MAPs and Common Memory nodes can all be connected to Hi-Bar in any configuration as shown in Figure 4. Each input or output

port sustains a yielded data payload of 1.4GB/sec for an aggregate yielded bisection data bandwidth of 22.4GB/sec per 16 ports. Port-to-port latency is 180ns with Single Error Correction and Double Error Detection (SECDED) implemented on each port.

Hi-Bar switches also can be interconnected in multitier configurations, allowing two tiers to support 256 nodes. Each Hi-Bar switch is housed in a 2U-high, 19-inch wide rackmountable chassis, along with its power supplies and cooling solution, for easy inclusion into rack-based servers.



[5]

Figure 5. Block Diagram of SRC-6 with Hi-Bar Switch

SRC servers that use the Hi-Bar crossbar switch interconnect can incorporate Common Memory nodes in addition to microprocessors and MAPs. Each of these Common Memory nodes contains an intelligent DMA controller and up to 8GBs of DDR SDRAM. The SRC-6 MAPs, SNAPs and Common Memory node (CM) support 64-bit virtual addressing of all memory in the system, allowing a single flat address space to be used within applications. Each node sustains memory reads and writes with 1.4GB/sec of yielded data payload bandwidth.

The CM's intelligent DMA controller is capable of performing complex DMA functions such as data packing, strided access and scatter/gather to maximize the efficient use of the system interconnect bandwidth. Interconnect efficiencies more than ten times greater than a cache-based microprocessor using the same interconnect are common for these operations.

In addition, SRC Common Memory nodes have dedicated semaphore circuitry that also is accessible by all MAP processors and microprocessors for synchronization.

Programming Model for Reconfigurable Computing

Traditionally, the programming model for RC has been one of hardware design. Given that the tools required for the underlying FPGA technology of RC are all logic design tools from the Electronic Design Automation industry, there really has not been a programming environment recognizable to a software developer. The tools have supported Hardware Definition Languages (HDLs) such as Verilog, VHDL and Schematic Capture.

With the introduction of system-on-a-chip (SOC) technology and the complexity associated with hardware definition of such complexity, high-level languages have begun to be available. Java and C-like languages are becoming more common for use in programming RC chips. This is a significant step forward but continues to require quite a leap by application programmers.

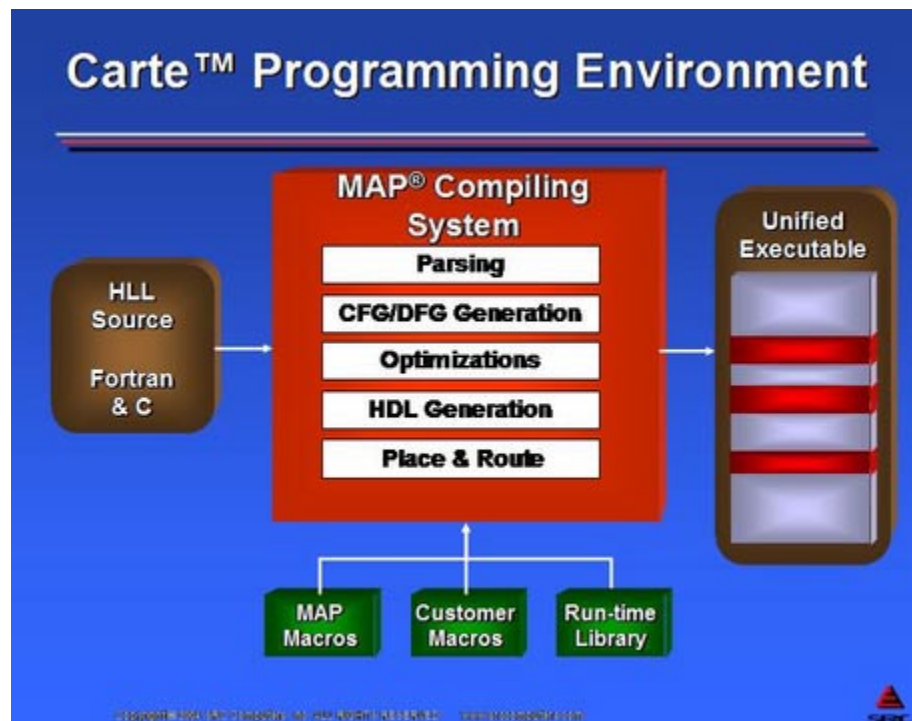
The SRC programming model is the traditional software development model where C and Fortran are used to program the MAP processor, and any language capable of linking with the runtime libraries (written in C) can be compiled and run on the microprocessor portion of the system.

The SRC Carte programming environment was created with the design assumption that application programmers would be writing and porting applications to the RC platform. Therefore, the standard development strategies of design, code in high-level languages (HLLs), compile, debug via standard debugger, edit code, recompile and so on, until correct, are used to develop for the SRC-6 system. Only when the application runs correctly in a microprocessor environment is the application recompiled and targeted for the DEL processor, MAP.

Compiling to hardware in an RC system requires two compilation steps that are quite foreign to programming for an instruction processor. The output of the HLL compiler must be a hardware definition language. In Carte, the output either is Verilog or Electronic Design Interchange Format (EDIF). EDIF files are the hardware definition object files that define the circuits that will be implemented in the RC chips. If Verilog is generated, then that HDL must be synthesized to EDIF using a Verilog compiler such as Synplify from Synplcity.

A final step, place and route, takes the collection of EDIF files and creates the physical layout of the circuits on the RC chip. The output files for this process are a configuration bitstream, which can be loaded into an FPGA to create the hardware representation of the algorithm being programming into the RC processor.

The Carte programming environment performs the compilation from C or Fortran to bitstream for the FPGA without programmer involvement. It further compiles the codes targeted to microprocessors into objects modules. The final step for Carte is the creation of a unified executable that incorporates the microprocessor object modules, the MAP bitstreams, and all of the required runtime libraries into a single Linux executable file. Figures 6 and 7 present the Carte compilation process.



[6]

Figure 6. Carte Programming Environment

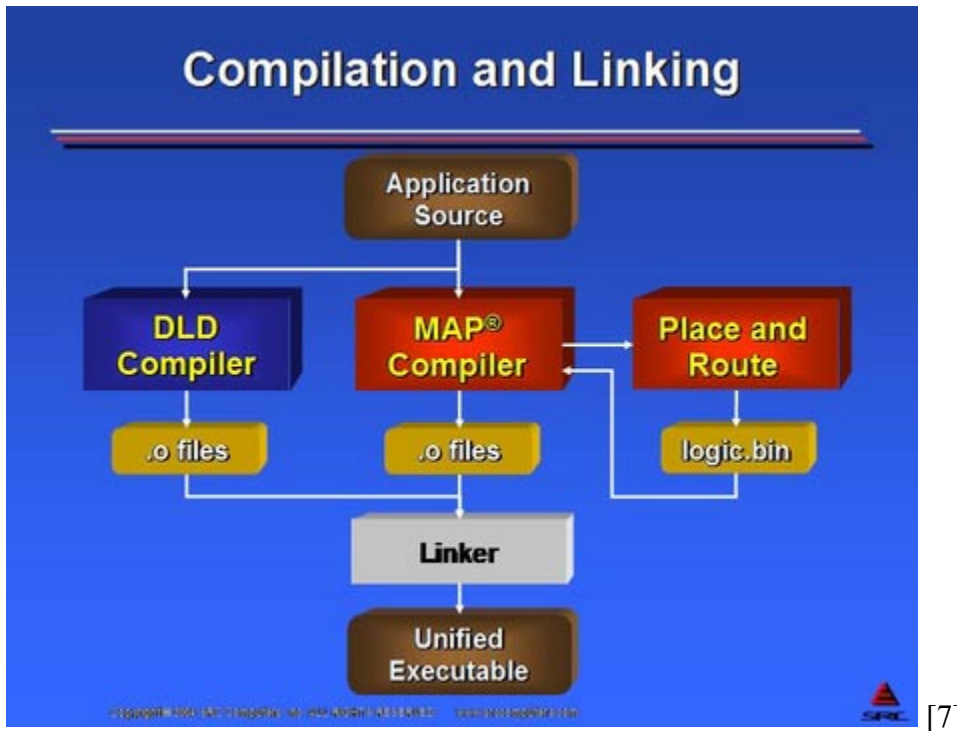


Figure 7. Carte Compilation

Open-Source Hardware Opportunity

Linux has led the way and benefited greatly from the Open Source movement where a large and dedicated group of software developers has created, modified and improved the Linux kernel and OS at a rate, quality and level of innovation that could not be matched by the work of a single commercial organization. Reconfigurable computing has the potential of enabling such innovation and technical advances in hardware design. Much of this article is spent explaining the concept of application programmers writing code and using standard programming methods to create application-specific hardware without requiring knowledge of hardware design. However, in RC the building blocks of the generated hardware created by application programmers is the functional unit. Functional units are basic computational units such as adders, floating-point multipliers or trigonometric functions. Functional units also can be specialty high-performance units, like triple DES functions, or nonstandard precision arithmetic units, such as 24-bit IEEE floating-point operators.

Functional units are created by logic designers. RC compilers, such as SRC's Carte MAP compiler, are capable of allowing customer-supplied functional units to be added to the standard set of operations supported by the compiler. When new and novel functional units are made available to application programmers, an even higher level of performance can be achieved.

It is in the creation and sharing of innovative hardware designs for functional units where an Open Hardware movement could bring substantial advances to computational science. The innovation and productivity seen in the open-source arena could be replicated as Open Hardware.

RC provides a vehicle for many more creative designers to create new and novel hardware that can be

used by application developers. Through groups like Opencores.org, functional unit design can be shared and improved upon. The significant advances seen in the computational sciences, due to open-source software, easily could be seen through a movement focused on open hardware as well.

Code Example

To show the performance advantage of a DEL processor, a string-matching example is presented. The code for these examples is available on the *Linux Journal* FTP site-see the on-line Resources. This example came from the Web site of Christian Charras and Thierry Lecroq, referenced by NIST Dictionary of Algorithms and Data Structures. For comparison, the Brute Force and Boyer-Moore string-matching algorithms are implemented for the 2.8GHz Intel Xeon via Intel's C++ 8.0 compiler for Linux. The Brute Force algorithm is implemented for SRC's system using the Carte 1.8 Programming Environment. The Brute Force algorithm is a straightforward character-by-character comparison between a pattern and a text string. The Boyer-Moore is considered the most efficient string-matching algorithm. The example takes a randomly generated 20MB text string and searches for six and ten randomly generated patterns. Compilations are done with a -O3 optimization setting, and performance comparisons are shown in Table 1. Adding four additional search patterns to the test increases the microprocessor times but has no impact on the MAP execution times due to the pipelined logic. Though the Xeon runs at 2.8GHz, and the MAP runs at 100MHz, the parallelism seen in DEL can achieve a 99× performance advantage in MAP. This example required 60% of one FPGA in the MAP. A two-chip compile would deliver over 200× performance.

Table 1. String-Matching Performance

Implementation	Text Size	Patterns	Search Time	Speedup
Brute Force (Xeon)	20MB	6	0.827 sec	1.00×
Boyer-Moore (Xeon)	20MB	6	0.597 sec	1.38×
Brute Force (MAP)	20MB	6	0.0143 sec	57.75×
Brute Force (Xeon)	20MB	10	1.398 sec	1.00×
Boyer-Moore (Xeon)	20MB	10	1.051	1.33×
Brute Force (MAP)	20MB	10	0.0141 sec	98.81×

To demonstrate the impact of adding additional computation into a pipelined loop, and the ability to introduce custom functional units, a second performance comparison is done in which a DES-encrypted string is passed to the search routine. The string must be decrypted prior to searching. In the case of the MAP implementation, a DES pipelined functional unit is introduced. The Verilog definition was obtained from Opencores.org and introduced into the search loop. Because the loop is pipelined, it continues to deliver a set of results per clock cycle. Therefore, the elapsed time for the 20MB text search, including a DES decryption, is unchanged from the search alone. This leads to a very dramatic 232× speedup over the microprocessor implementation. The ten-pattern MAP example uses only 74% of an FPGA, so a two-chip compile for the MAP would yield 460×.

Table 2. Performance for Searching an Encrypted String

Implementation	Text Size	Patterns	Search Time	Speedup
DES-Brute Force (Xeon)	20MB	6	2.77 sec	1.00×

DES-Boyer-Moor (Xeon)	20MB	6	2.63 sec	1.05×
DES- Brute Force (MAP)	20MB	6	0.0143 sec	193.09×
DES-Brute Force (Xeon)	20MB	10	3.31 sec	1.00×
DES-Boyer-Moor (Xeon)	20MB	10	3.11 sec	1.06×
DES- Brute Force (MAP)	20MB	10	0.0143 sec	231.76×

In the case of DES implemented on the Xeons, the code is an optimized code by Stuart Levy at Minnesota Supercomputer Center.

Conclusion

This article has explained reconfigurable computing, shown examples of the methods and the results that can be achieved. Significant performance gains can be demonstrated. In the present, RC has much to contribute to computational science, but the future holds advances well beyond the Moore's Law gains experienced in the world of microprocessors. RC is accessible to today's programmers using a familiar programming model and provides the framework within which a larger population of hardware designers can have an impact on high-performance computation through open-source creativity and productivity.

RC has been a long time in coming, but the enabling software and hardware technology has set the stage for RC to become part of every computer, from embedded processor to Peta-Scale supercomputer.

Resources for this article: www.linuxjournal.com/article/7867 [8].

Dan Poznanovic (poz@srccomp.com [9]) is VP Software Development at SRC Computers, Inc., and has been involved in the high-performance computing world since initially joining Cray Research, Inc., in 1987.

Links

[1] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f1.png>

[2] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f2.png>

[3] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f3.png>

[4] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f4.png>

[5] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f5.png>

[6] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f6.png>

[7] <http://www.linuxjournal.com//articles/lj/0129/7731/7731f7.png>

[8] <http://www.linuxjournal.com//article/7867>

[9] <http://www.linuxjournal.com/mailto:poz@srccomp.com>

Source URL: <http://www.linuxjournal.com/article/7731>