Elemental Computing with the Element CXI ECA

Peter Athanas
Virginia Tech
Department of ECE
Blacksburg, Virginia
Peter.Athanas@VT.edu

Bruce Gladstone
Director of Early Access Programs
Element CXI
Milpitas, California
Bruce.Gladstone@ElementCXI.com

Abstract: The Element CXI *Elemental Computing Architecture* (ECA) is a scalable, fully programmable platform for parallel and distributed processing. The architecture is comprised of a dynamically configurable fabric of heterogeneous *elements*—or dataflow engines—that time-share operations in a token-based data-driven flow. Computationally intensive tasks are distributed across elements for maximum speed and parallelism, while simple tasks time-share elements. This paper presents an overview of the Elemental Computing paradigm and contrasts it to contemporary FPGA-based reconfigurable computing flows.

Introduction

For two decades now, the field of reconfigurable computing has offered a powerful model for high-performance embedded computing. The foundation of this model is ability to dynamically customize the organization, functionality, and connectivity of an underlying computational platform based upon the computation at hand. In doing so, many benefits can be achieved. Higher computational throughput is attained through the realization of custom computational pipelines. Lower power requirements are in theory achieved through reduced circuit activity. Since a computation is customized to a particular application within a reconfigurable fabric, there is less overhead associated with control logic as one would find in a general-purpose processor. In addition, reconfigurable computing also offers other advantages to certain classes of problems, such as size and weight reduction, and accelerated deployment times.

Until now, FPGAs have been the primary vehicle for reconfigurable computing systems. Using a relatively fine-grain RAM-based architecture, FPGAs exhibit a high degree of flexibility. This flexibility, however, comes at a high price in terms of computational density and power efficiency. In theory, the dynamic nature of FPGAs can be exploited to achieve greatly enhanced computational density by "swapping out" otherwise idle circuitry capabilities. In practice, however, FPGAs are used primarily as "rapid prototyping" platforms, where the computational task is statically configured for the lifetime of the application, regardless of its usage duty cycle. There are several reasons for this including (a) the reconfiguration bandwidth in contemporary FPGAs is relatively slow (several thousands to millions of clock cycles), (b) the *use-model* for dynamic behavior is poorly defined, and (c) run-time reconfiguration is not well supported by the vendors in terms of tools and documentation. Despite two decades of research and development, there are only but a handful of mainstream FPGA applications that exploit the full benefits of reconfigurable computing.

The Element CXI *Elemental Computing Architecture* (ECA) is a reconfigurable computing paradigm that is built on a data-flow computational model that facilitates the creation of computationally dense and power efficient applications. ECA devices can be applied to many applications associated with communication systems, networking, and multimedia applications. Furthermore, the combination of power efficiency, flexibility, and fast real-time reconfiguration make the ECA well suited for software-defined radio systems. This paper presents an overview of the elemental computing concepts and computational model.

ECA Architecture

The Element CXI ECA is composed of a collection of heterogeneous *elements*, or computational building blocks, that can be dynamically chained to form specific computing structures on demand. Elements can be considered to be coarse-grain computational units in that they provide functions ranging from complex multipliers to fully capable RISC processors. The architecture favors signal processing computations, yet is sufficiently flexible to do well on applications beyond this focus. Elements communicate through 16-bit buses, yet individual buses can be combined to provide 32-bit and 64-bit operations. Despite the emphasis on word-wide computational pathways, the ECA does extremely well in single bit-wide computations and operations. The coarser granularity provides a substantial boost in power efficiency and computational density when compared to FPGAs. The advantages of coarser-grain architectures have been clearly established in the literature and by other devices in the marketplace [1].

The ECA provides a number of capabilities that further distinguish it from conventional DSPs, GPUs, and FPGAs. Each element in the ECA fabric can possess several different *contexts*. Each context contains a full configuration of the operation of the element. This allows the computational core of an element to be shared with (or bound to) different portions of the running application. For example, one context for an element could be

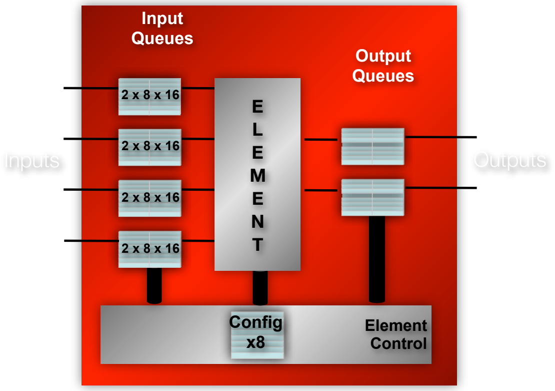


Figure 1: All ECA elements are wrapped to provide input and output queues and a multi-context configuration controller.

responsible for a FIR filter tap computation, while another context performs a polynomial computation. An element can switch from one context to another transparently in a single clock cycle. The mechanism for deciding to switch from one context to another is most interesting. Unlike FPGAs, where the algorithm designer must arduously and manually floorplan and time-slice a multi-context application, the decision to switch the context an element within an ECA is performed automatically at run-time with no design-time planning or intervention. Furthermore, the decision made by one element to context switch is independent of the other elements. In summary, each element within an ECA array independently decides on a cycle-by-cycle basis, what the next computation shall be. This is a key capability that folds elegantly into the computational model, presented in the next section.

The concept of incorporating context switching within a reconfigurable device is instrumental in improving both computational density and power consumption. In regards to computational density, an otherwise idle operator can be swapped out in favor of a pending computation, vastly increasing the overall number of operations performed in a device per clock cycle. Non-active contexts remain completely static eliminating all unnecessary activity, resulting in significant power savings.

Elements are interconnected through a rich hierarchical switching network. The network is dynamic in that the connectivity between elements can change on a clock-by-clock basis. Elemental interconnect is word-based, and includes a broadcast capability so that a single element can transmit to multiple destinations. The flexible interconnect structure provides the close coupling of resources needed by algorithms sensitive to feedback or to next-iteration latencies [2]. Word-wide organization of the device connectivity has a dramatic impact on reducing power consumption when compared to FPGAs. Furthermore, connectivity is fully deterministic, reducing the need for doing complex compile-time timing closures. All designs operate at the full rated speed of the ECA device.

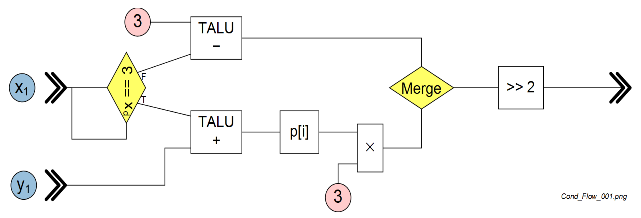


Figure 2: An illustration of how control-flow constructs are readily implemented in the ECA architecture. Here, an IF-Then-Else construct is formed satisfying the conditional:
*if (x==3) { o = p[x+y] *3; } else { o = x-3;}; o >>= 2;*

In addition to computational elements, there are large amounts of distributed memory throughout the ECA. These memory elements are instrumental in keeping concurrent operations primed, and result in a huge cross-sectional bandwidth in the ECA fabric. The ECA memory elements also eliminate the need to use extraneous resources to implement common storage functions. Operations such as multi-dimensional scan patterns, scatter/gather operations, FIFOs, and rendezvous synchronization points are all inherent in the ECA fabric. Distributing memory close to the elements adds a degree of fault tolerance. Having one distributed memory fail leaves others that can (dynamically) step in and maintain operations.

Computing Model

The *computational mode*l of a device is the central theme that binds software programmability to the hardware's capabilities. In the ECA, an enhanced CSP dataflow model[1] is used [3]. Abstractly, valid data are tagged with tokens as they move through the ECA fabric. All inputs of all contexts of all elements in the system feature input queues. Similarly, all contexts of all elements feature queues for outgoing data. Tokens for a given computation flow into input queues and remain there until an element can *fire* and *consume* the tokens. A given context for an element will fire if and only if (a) all significant inputs for the context have tokens available, and (b) there is room in the context's output queue for a newly produced token(s). This model has a number of distinct advantages:

1. The movement of data are governed by both forward flow control (space availability in input queues) and backpressure (space availability in output queues). Because of this, data are never lost in the system due to overflow or under-run conditions.

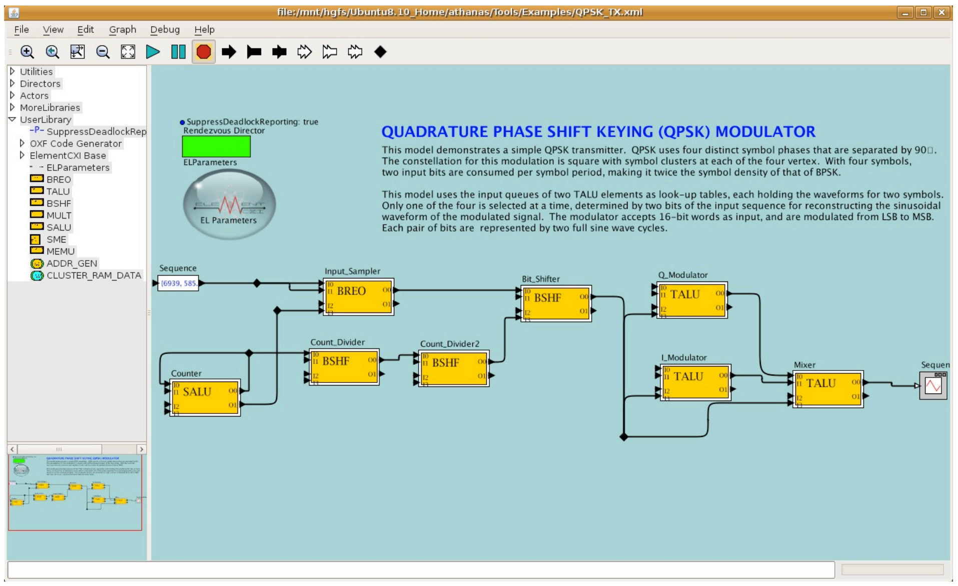
[1] Communicating Sequential Processes (CSP) model allows a system to be described in terms of component processes that operate independently, and interact with each other solely through message-passing communication. The CSP model has a sound mathematical foundation along with a wealth of contemporary tools and development environments.

2. Data rate conversions (e.g., as in multi-rate signal processing) are performed implicitly in the ECA fabric requiring no additional effort by the algorithm designer.
3. If the inputs for a given context are all available at the same time, and there is room for the output token there is no queuing delay or overhead, and the element fires instantly. Similarly, an output queue can be bypassed if the destination is free. These are important factors in pipeline computational efficiency.

An important consequence of this and the core computing model is that each context "knows" if it can fire or not based upon the above conditions. Because of this, the decision to context switch an element is straightforward, and is easily folded into the ECA hardware fabric.

Another important consequence of this is the concept of virtualization. The ECA hardware will have only a finite number of elements within a given device; however, the designer does not need to be burdened with the physical limitations of the device. Instead, applications can be created for the ECA that consist of many more times the number of logical elements than there are physical elements in the device. The hardware, not the designer, will manage resource sharing at run-time, not at compile-time. Furthermore, if the contexts within the device are depleted, there are several RISC processor elements distributed throughout the device that can also intervene in the run-time configuration of the device in a distributed random-access manner. Together, these factors will likely boost productivity significantly for dense complex designs.

The computational core of most signal processing kernels are readily expressed as dataflow computations, which makes the mapping to ECA primitives easy. Most signal processing tasks also have some degree of control-flow aspects to them, which are difficult to express in conventional dataflow primitives. The architects of the ECA foresaw this issue and crafted the element repertoire to readily express common control-flow constructs. Conditionals (if-then-else) and looping constructs (for, do, while) are easily instanced in an application (refer to Figure 2). Furthermore, complex control mechanisms can be easily implemented in the RISC-based elements.


QUADRATURE PHASE SHIFT KEYING (QPSK) MODULATOR

This model demonstrates a simple QPSK transmitter. QPSK uses four distinct symbol phases that are separated by 90□. The constellation for this modulation is square with symbol clusters at each of the four vertex. With four symbols, two input bits are consumed per symbol period, making it twice the symbol density of that of BPSK.

This model uses the input queues of two TALU elements as look-up tables, each holding the waveforms for two symbols. Only one of the four is selected at a time, determined by two bits of the input sequence for reconstructing the sinusoidal waveform of the modulated signal. The modulator accepts 16-bit words as input, and are modulated from LSB to MSB. Each pair of bits are represented by two full sine wave cycles.

SuppressDeadlockReporting: true

Figure 3: An example ECA application (a QPSK transmitter) built from elements (yellow boxes) expressed as a data flow graph within the Ptolemy design environment.

Conclusion

Elemental computing is a parallel, distributed, data-flow paradigm. Data transfers in every stage in a task are reliable and inherently controlled within the ECA fabric, where the complexity is hidden from the user. As a consequence, portions of an application can be relocated seamlessly throughout the fabric. Unlike FPGAs, applications and pieces of applications can be mapped and remapped on a clock cycle by clock cycle basis if needed.

For academicians, the ECA architecture provides a means of exploring many interesting research issues that cannot otherwise be investigated with other technologies, such as FPGAs and DSPs. Some of these issues include (a) true run-time reconfiguration and hardware virtualization, (b) resiliency and fault recovery, and (c) hardware operating systems. All of these factors will play an important role in the future of high-performance embedded computation, and will encourage designers to rethink how contemporary media devices, radios, and networks are made.

References

[1] R. Hartenstein, “Coarse grain reconfigurable architecture,” Proceedings of the 2001 Asia and South Pacific Design Automation Conference, Yokohama, Japan, pp. 564 - 570 , 2001
[2] S. Kelem, B. Box, S. Wasson, R. Plunkett, J. Hassoun, C. Phillips, “An Elemental Computing Architecture for SD Radio,” Proc. of 2007 Software Defined Radio Technical Conference and Product Exposition, Denver, Colorado, November. 2007.
[3] C. Hoare, "Communicating sequential processes," *Communications of the ACM* **21** (8): 666–677, 1978.
[4] C. Maxfield, “Dynamically-reconfigurable Elemental Computing Arrays (ECAs),” http://www.pldesignline.com/202803397, November 2007.