Re-Configurable Computing in Wireless

Bill Salefski
Chameleon Systems, Inc.
161 Nortech Parkway
San Jose, CA 95134 USA
+1 408 240 3400
salefski@cmln.com

Levent Caglar
Chameleon Systems, Inc.
161 Nortech Parkway
San Jose, CA 95134 USA
+1 408 240 3401
levent@cmln.com

Abstract

Wireless communications requires a new approach to implement the algorithms for new standards. The computational demands of these standards are outstripping the ability of traditional signal processors, and standards are changing too quickly for traditional hardware implementation. In this paper we outline how reconfigurable processing can meet the needs for wireless base station design while providing the programmability to allow not just field upgrades as standards evolve, but also to adapt to much more dynamic factors in the environment such as traffic mix and the weather. We outline how a designer works with the Chameleon reconfigurable processor from algorithm design to prototyping on a development module.

1. Introduction

System designers require new architectures to meet the signal processing demands for today's wireless applications. There are at least three design dilemmas driving this need for change: the gap between what traditional processing architectures can provide and what wireless equipment needs for signal processing, the dynamics of wireless communication standards, and the realities of building large system on chip (SOC) solutions.

Designers looking to extract transmission capacity from a relatively limited amount of spectrum are placing rapidly increasing demands on the required signal processing to support new standards. Jan Rabey showed in [1] how the computing requirements for new standards are exceeding the forecast processing capacity available through semiconductor process improvements.

Wireless communication standards evolve quickly and demands for new features and services within these standards arise much more rapidly than in the past. This dynamic environment drives the equipment manufacturers towards programmable solutions that they can update in the field. The ideal is a base station platform that service providers can deploy and then upgrade in the field with only software changes over its operational life.

Mark Horowitz outlined in [2] the realities of CMOS process technology, pointing the direction for the use of specific architecture requirements that match process realities. In his presentation, Horowitz shows how *modular computers*, arrays of computational elements with regular interconnect and localized programming, provide substantial value when implemented in the new technologies.

Reconfigurable computing has gained significant attention of late as a commercially viable and available alternative to conventional processing schemes. Chameleon's Reconfigurable Communications Processor (RCP) architecture delivers an array of computational elements with a programmable interconnect. This allows designers to essentially create a custom-designed data path and memory structure for their application that they can instantly alter to implement the desired algorithm.

This paper provides an overview of the Chameleon RCP, how designers use it, and its performance relative to other solutions. Section 2 examines the major architecture features of the RCP. Section 3 shows how to structure algorithms to efficiently execute on the processor. Section 4 outlines the specifics of the design methodology to program the processor. Section 5 compares the performance of the processor to existing DSP, FPGA, and ASIC approaches.

2. Chameleon Architecture

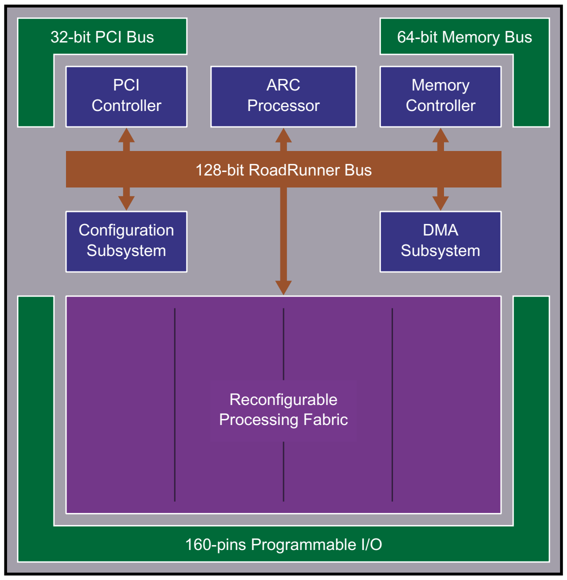

Figure 1. Chameleon RCP

The RCP architecture (Figure 1) comprises three major subsystems: the Reconfigurable Processing Fabric (RPF) which contains the bulk of the computational elements, the programmable input and output (PIO) banks to stream the data through the fabric, and an embedded processor subsystem. These system modules are linked with a high-performance 128-bit bus that provides the bandwidth to move data among them.

2.1 Reconfigurable Processing Fabric

The RPF consists of data processing units, local storage, and the interconnect structure among these elements. The RPF's architectural objective is to match the fast, distributed data processing with a fast, localized memory. Data are streamed into the local memories, processed, and streamed out.

The RPF for the CS2112 is arranged in a hierarchy as shown in Figure 2. The top-level of the RPF is divided into four *slices*, each of which can be reconfigured independently of the other slices. Each slice is subdivided into three *tiles*, and each tile consists of seven 32-bit datapath units (DPU), two multipliers (MPU), four local-store memories (LSM), and a Control Logic Unit (CLU).

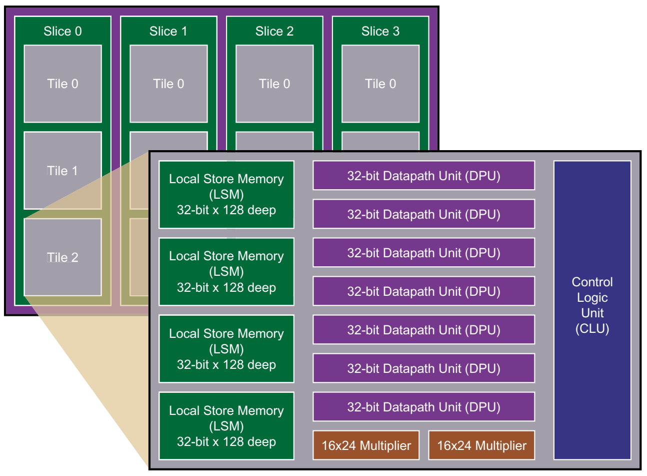

Figure 2. Reconfigurable Processing Fabric

2.1.1 *Processing Elements*

The DPU shown in Figure 3 is the fundamental computational element in the fabric. It has a basic word length of 32 bits, but has special operations that allow it to operate in SIMD fashion on four 8-bit data streams or two 16-bit data streams.

The core of the DPU is the 32-bit *Operator* that performs arithmetic and logical operations. Operations include ADD, ADD16 (two 16-bit additions performed in parallel), MAX, SADD (saturating addition), and so forth. One branch of the operand data path contains a barrel shifter for performing shifting, word swapping, byte swapping, or word duplication on an operand. All registers have conditional enables to allow pipelining and to allow the data in registers to be preserved between reconfigurations or initialized to constant values.

The MPUs can perform 16x24-bit or 16x16-bit single-cycle multiplications. The multipliers have a similar input and output configuration to the DPUs, but only perform multiplication.

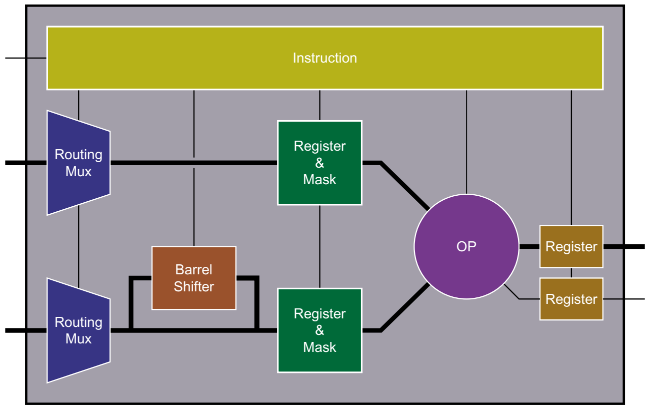

Figure 3. Datapath Unit

2.1.2 *Local-Store Memory*

The LSMs form the basic building blocks for the localized memory architecture. Each LSM is a multi-ported, 32-bit by 128-word RAM. Through programming, LSMs can be assembled into different memory configurations to match the data stream they are capturing or sourcing. For example, two LSMs can be programmed to form a wide-word memory structure providing 64-bit by 128-word RAM, or the same two can be assembled to form 32-bit by 256-word RAM. Data in the LSM are retained during these configuration changes.

A common use of this memory reconfigurability is to load the data very quickly with wide words and then process it through the DPUs and MPUs as narrower data. For example, the designer can load the four LSMs in a tile as a 128-bit by 128-word RAM, and then reconfigure them into 16-bit buffers to feed DPUs for the actual signal processing.

2.1.3 *Control Logic*

Each DPU is programmed with eight user-definable instructions stored in the *Instruction* memory. Each instruction specifies the complete configuration for the DPU. This includes the input and output routing, shifting, masking, register enables, LSM read and write instructions, flag generation, and the DPU Operator. So each instruction creates a custom datapath, data flow, and memory architecture for the particular operation.

The Control Logic Unit (CLU) is shown in Figure 4. The CLU directly implements a finite-state machine to select the DPU and MPU instructions stored in the instruction memory.

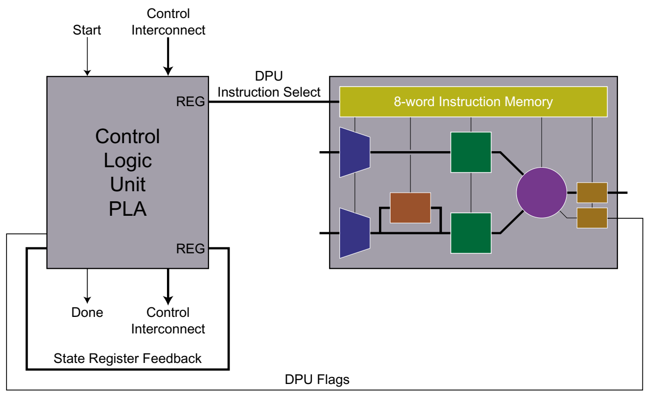

Figure 4. Control Logic Unit Connection

The CLU consists of a PLA containing the next-state and combinational functions, state registers, and routing multiplexers. The PLA inputs come from state bits, the DPU flags, and control bits sent from CLUs in adjacent tiles. The state bits are used to index the instruction memory to select the instruction for the DPU to perform. The PLA also outputs control information to adjacent CLUs in other tiles. The *start* and *done* bits are used by convention to initiate fabric processing and to signal when the processing is complete.

2.2 Dynamic Interconnect

The interconnection of DPUs consists of 32-bit data buses and individual control lines. These busses do not have a signaling protocol; they are direct connections among the DPUs chosen for interconnection. Nearby DPUs are connected to each other in a full crossbar connection. More distant DPUs are connected with routing with a one-clock pipeline delay. Routing multiplexers control the routing selection for the DPUs so the interconnect can change on a clock-by-clock basis to implement the optimal dataflow for the algorithm.

2.3 Programmable I/O

The RCP has four banks of 40 programmable input and output pins that provide the bandwidth for streaming data through the reconfigurable fabric. The PIOs can clock at the system clock rate, providing a raw data bandwidth of about 2.0 Gbits/sec to enable high-speed, data-streaming applications.

The PIOs can be programmed to interface to a variety of external data sources, including SRAM, A/D converters, and FPGAs.

2.4 Embedded Processor Subsystem

The embedded processor subsystem provides the components for a complete system, and provide the management for reconfiguring and streaming data for the fabric. The processor is an ARC processor with minor changes to tailor it for wireless applications. It has an 8K-instruction cache and an 8K-data memory. The system also contains a 32-bit PCI interface, a DMA, and a 64-bit memory controller to interface to external SDRAM, SSRAM, and flash memories.

A unique aspect of the RCP is that all storage elements in the fabric are mapped to the ARC address space for both reading and writing. This enables the designer to use the ARC to control the fabric during processing, and gives the designer complete visibility into the fabric for system debug and bring up.

2.5 Reconfiguration

2.5.1 System Reconfiguration

The reconfiguration system for the entire RCP consists of a *configuration controller* and two configuration planes. The *active* configuration actually controls the fabric, and the *background* plane is used to hold another configuration.

The controller can load a new configuration into the background plane while the fabric is running with the active plan. To program all four slices, less than 50,000 bits are required, so a new configuration can be loaded in less than 3 microseconds.

The entire system can be reconfigured in one clock cycle by switching the data from the background plane to the active plane. Each slice in the RCP can be reconfigured independently, although most applications configure all four slices as a group.

The reconfiguration changes the contents of the CLU, but data in the LSMs is retained. Therefore, data can be stored in LSMs from one configuration to be used as input data in the next configuration.

2.5.2 Instruction-Based Reconfiguration

Since the DPUs and the Dynamic Interconnect are both completely specified in the instruction memory and controlled by the CLU, it is possible to have different configurations on a clock-by-clock basis built into the programming. Designers use this level of reconfiguration to fine-tune algorithms or switch to different processing based on data values.

3. Design Methodology

Chameleon's RCP serves as a reconfigurable platform that enables customers to design, debug and deploy their proprietary algorithms. Therefore, the design methodology and tools Chameleon provides to enable its customers to program the RCP are critically important for successful implementations.

There are two key aspects of the Chameleon design methodology that are unique for the RCP. The first requires the customer to consider the massive amount of parallelism and programmability that the RCP makes available to the designer. The second introduces the potential of effectively using the power of reconfiguration as an aspect of the design process. These are totally new design parameters, but their effective utilization can greatly increase the capacity of the algorithms implemented on the RCP over traditional programmable approaches.

The design process follows three basic steps. The first is to define the data flow through the system, the second is to map the operations to the architecture elements, and the last is to enter the design into the RCP programming tools.

3.1 Dataflow Design

The designer first encounters the power and flexibility provided by the RCP architecture when designing the dataflow through an RCP-based. Some traditional design constraints necessary on other programmable systems are relaxed when designing on the RCP. The steps basically include establishing the data set size, structuring the parallelism in the algorithms, and determining the control needed to manage the data flow.

3.1.1 Data Set

The first step for the designer is to determine the data set size for the RCP. The most effective method is to use streaming dataflow, i.e., to consider the input as an unbounded array of data to be processed. The PIOs can be configured to directly load the data into the LSMs in the fabric and transfer the data out when the processing is complete. Although streaming dataflow is the best match for the RCP, frame-based data sets also work well, with larger sets being more efficient.

3.1.2 Parallelism

The next step is to map the algorithm onto the array of DPUs. The amount of computing resources and the flexibility for their connection allows the designer to design the optimal mix of *task-level parallelism* and *instruction-level parallelism* for the particular algorithm. The designer is not constrained by fixed data paths or relatively small numbers of processing elements.

The designer extracts the instruction-level parallelism to determine the number of operations that can execute in parallel for a particular processing task, such as FIR taps computed at the same time. Task-level parallelism is extracted to determine processes that can execute in parallel. Since these are both programmable, it is possible for the designer to develop several processes that run in parallel to execute in the same number of clock cycles, so they start and finish in unison.

To effectively use the quantity and flexibility of the parallel operations available on the RCP, the designer generally approaches the algorithm design from a signal-flow graph or C code that is structured like a signal-flow graph. This is best achieved by taking the original signal flow graph or dataflow specification of the algorithm from tools like Mathworks' MatLab, Cadence's VCC, or Synopsys's Concentric System Studio.

Because of the number of computational resources made available by the RCP, it is sometimes more efficient for the designer to compute results that are redundant or to speculatively compute results even if they may not be needed. For example, it may be more efficient to compute the same result in two different parts of the algorithm rather than trying to compute it once and then communicate the result to both places. It can also be more efficient to compute both outcomes of a conditional computation and then select the result at the end.

3.1.3 *Control*

To minimize the disruption on the dataflow and processing pipelines, the design should reduce the amount of control and feedback. Several designs have been done in which an inherently control-oriented algorithm was restructured to have a steady state of dataflow. For example, one particular algorithm required the RCP to accept a data stream that contained irregularly spaced invalid data. When the algorithm was implemented on a DSP, a simple conditional statement was used to exclude the invalid data from the data set. Such an approach would have lead to an irregular pipeline and large control. Instead, the designer associated a 'valid' bit with each sample as it was processed. The data path and streams stayed constant, and simple local control was employed to filter out the invalid data.

3.2 Architecture Mapping

The next task is to develop partitioning, schedules, and processing assignments that keep the hardware resources busy and bind the operations to the DPUs. This is a scheduling problem that involves the fundamental algorithm, communication latencies, and resource availability.

3.2.1 *Pipelining*

When mapping to the hardware, the pipelines delays can cause an algorithm to have some dead cycles. Since the RCP is user programmable, the designer can still make use of these cycles for other data. One of the common approaches is to interleave other data streams for the same or similar algorithm through the datapath. Kirin Bondalapati shows in [3] the detailed analysis and implementation of this technique.

3.2.2 *Reconfiguration as Design Component*

An aspect of design unique to the RCP is using reconfiguration as a part of the algorithm implementation. The UC Berkeley SCORE project under Prof. John Wawrzynek is extensively researching this topic [4]. Chameleon has also done some analysis on how this can effectively increase the hardware utilization for wireless applications by reconfiguring the hardware as part of the algorithm implementation.

For some frame-based applications, the frame requires several distinct processing steps. For example, the 1250 microsecond power control group in the CDMA protocol employs specific processing algorithms for code generation, demodulation, searching for multi-path components, and searching for new users attempting to access the system. Rather than having separate processing units, Figure 5 shows how the RCP's reconfigurability allows the designer to create optimized data paths for each processing stage of the power control cycle

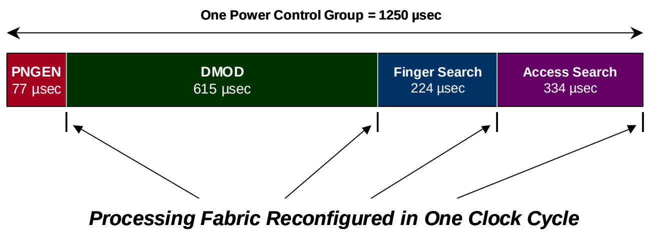
One Power Control Group = 1250 µsec
PNGEN 77 µsec
DMOD 615 µsec
Finger Search 224 µsec
Access Search 334 µsec
Processing Fabric Reconfigured in One Clock Cycle

Figure 5. Example Reconfiguration Timeline

Another design aspect for using reconfiguration is to vary the algorithm depending on environmental considerations. For example, if the weather is rainy the designer will want to include more multi-path components in the demodulation algorithm.

The designer also has the ability to reprogram completely different algorithms based on the traffic mix. If the algorithms are similar enough to permit the board signal paths to be reused, the RCP can provide rapid switching and upgrades in the field.

3.3 Comparison to Other Approaches

Several of the key design steps for the RCP are somewhat similar to design methodologies used for DSP and ASICs. However, there are some significant differences.

3.3.1 *FPGA and ASIC Design*

Generally speaking, ASIC and FPGA designers follow a methodology very similar to the methodology for the RCP. The major difference is that programming the RCP is completed when the DPU and control functions are specified and mapped onto the RCP. Using an FPGA or ASIC, the designer must continue to refine the design with the micro architectural details until they have a complete net list ready for synthesis, placement, and routing. Then they must perform the actual layout floor planning, timing closure, processor integration, and other steps to complete the design. In addition, the ASIC and FPGA design tools deal with much more complexity than the RCP tools need to, so their run times tend to be measured in hours, whereas RCP tool runtimes usually require a few minutes.

ASIC and FPGA providers are trying to simplify this latter process, but in the end the designer is responsible for insuring that the chip layout actually works. The RCP, in contrast, has all of these issues solved before it is delivered to the designer.

3.3.2 *DSP Design*

For DSP programmers who use assembly language on VLIW machines, the RCP design process can be similar. However, the RCP offers much more parallelism and allows the designer to specify the datapath and memory architecture for the particular application.

The first step for assembly programmers is to design the data flow. On a DSP, they partition the data set to fit into internal memory and use the DMA or serial ports to move data in and out of the chip. The RCP does have a DMA capability that can be used in a similar fashion. However, for increased bandwidth on the RCP, the designer can use the PIOs and structure the algorithm so it follows a streaming dataflow model.

The next step for DSP assembly programming is to find the best way to map to the parallel operations provided by the DSP's instruction set, and then modify the algorithm to better fit the DSP architecture. Conversely, the RCP offers many more resources that can be used in many different ways. For example, a TI C62 has two data paths, each with what might be considered to be three DPUs, and a multiplier. The challenge for the DSP designer using the RCP is to expand the amount of parallel operations to take advantage of the over 100 processing elements on the RCP.

DSPs are basically sequential processors with a limited amount of parallelism, so designers try to keep the pipelines full with essential computations. In contrast, efficient use of the RCP's extensive parallelism requires that a steady dataflow be set up and maintained. Thus, designers can sometimes implement algorithms much more efficiently on the RCP by utilizing the relatively abundant resource of DPUs to perform redundant operations and speculatively compute results. If this extra computing reduces branching, and therefore maintains a steady dataflow, then the resulting implementation will be much faster and more efficient than the corresponding DSP assembly code.

4. Design Tools

Chameleon design tools consist of tools to program the fabric and tools for the embedded processor. The fabric tools are Chameleon specific because of the unique nature of the fabric; the system tools are based on industry-standard compiler and library products.

4.1 Fabric Design

Fabric design methodology requires the designer to specify the programming for individual DPUs, and then the tools map, place, and route these processing elements onto the actual fabric. The current tools are based on Verilog as a specification language and a design methodology modeled after the general synthesis-based methodology used for ASIC and FPGA-based designs. Fabric design comprises four major steps:

To design the data path, the designer inserts library components that model the functionality of the DPUs. The designer specifies the operations, shifting patterns and other basic functional characteristics.

The control is specified as a general Verilog state machine. The tool takes the description and applies industry-standard optimization and mapping techniques to program the control resource.

Place-and-routing software maps the DPU and the control programming onto the actual RCP resources. Then the routing software generates the programming for the interconnect multiplexers. At the end of this phase, an actual bit stream for RCP programming is emitted.

If there are any failures in the synthesis process, the designer uses floor-planning tools to set and update constraints on the placement.

We are increasing the level of abstraction so this flow plugs into higher-level tools such as MatLab. The general techniques for scheduling and module generation are applicable for the RCP and promise to make RCP programming much easier.

4.2 Embedded Processor Design

Chameleon uses the industry-standard GNU-based tool chain to program the embedded processor subsystem. The generated code targets the ARC and the peripherals available on the RCP. Basic standard functions are provided, such as *stdio*, to allow designers to prototype software on a host processor and then migrate the software to the target ARC.

Chameleon provides a set of system libraries, called eBIOS™, that run on the ARC to control the fabric. These libraries enable the designer to load bit streams, initiate processing, communicate status to the fabric, and control the rest of the embedded processor system. The designer can use these functions directly, or they can use a set of *pragmas* and a pragma processor, whereby the fabric-based functions are invoked just like regular C function calls.

Typical fabric control functions performed on the ARC include initiating processing on the fabric, loading a configuration in the background plane, switching between the active and background planes, and scheduling the next fabric loads. Data movements are performed by scheduling DMAs and loading data into LSMs from the ARC. The designers also put command and control processes on the ARC to interface to the host processor for the system.

4.3 System Verification

To validate the design, the designer has the option of using a software simulator that emulates the entire chip, or to load the design onto an actual RCP that Chameleon provides on standard PCI-based evaluation board. This latter capability is feasible because the RCP is designed such that all storage elements are visible to agent programs running on the ARC. This enables the designer to actually evaluate the performance of the RCP with the test data coming from the PC host.

4.4 C-Based Design

A C-based design language will play a part in design for architecture like the RCP, but there are certain aspects that make these undesirable as the only high-level entry mechanism. The most significant is that the fundamental execution model of C is a sequential machine. Algorithms coded in C tend to have the parallelism obscured and use fixed size arrays.

By contrast, to extract the maximum capability from the RCP, the data must stream through the system instead of being processed in frames. This tends to map better to the reality of actual embedded systems where data comes directly from streaming devices such as antennae.

Control constructs such as loops obscure the inherent parallelism in algorithms. Paradoxically, designers spend a significant amount of design effort to code parallel algorithms efficiently in C, which then requires a significant amount of effort on the part of the compiler to reconstruct the original, parallel algorithm.

There is certainly a role for C in the design of smaller blocks, such as DPUs, where the language only deals with a few threads, and in system modeling and test benches. However, as a design-entry language for the system or algorithm description, there is much more to be gained by using higher-level descriptions based on dataflow or other inherently parallel semantics.

5. Performance Comparison

The RCP is currently implemented at 0.25 µm process with a 125 MHz clock. The core runs at 2.5V and the PIOs run at 3.3V. We use this as the basis for comparison with other approaches.

5.1 DSP Processor

Generally, for a wide class of algorithms, the much greater number of operators on the RCP allows the designer to take advantage of task and instruction-level parallelism. In a DSP, adding more processing units to the board or SOC, and then managing the control of their integration can achieve more task-level parallelism. The designer can increase instruction-level parallelism in a DSP-based algorithm by adding more execution units. However, these are alternatives that are only available with an ASIC or FPGA methodology using a configurable core.

We compared the RCP implementation of several key kernels to the equivalent function running on a TI TMS320C62-300 300MHz fixed-point processor. We used the TI data published on the TI web site at [7].

Kiran Bondalapati performs a rigorous analysis in [3] of a filter implementation on the RCP, on the TI DSP, and on other sequential processors.

For a 24-tap FIR filter with 16-bit data, the Chameleon RCP is able to compute 200 samples in 233 clock cycles—or 1.9 µs. From the TI benchmarks, the equivalent processing takes 11.0 µs; in this case the RCP is about 6 times faster.

For a 1024-point complex FFT or inverse FFT with 16-bit real and 16-bit imaginary data, the RCP can process one block of 1024 samples in 5320 clock cycles, or 42.6 µs. However, the RCP can hold 4 instances of this FFT running in parallel. If the data streams are properly staggered, the RCP can output one block of 1024 samples every 10 µs. From the TI benchmarks, the equivalent processing takes 68.7 µs. Including the benefit of pipelining on the RCP, the RCP is about 6 ½ times faster than the TI DSP.

5.2 FPGA

FPGAs may have similar performance compared to an RCP implementation. However, there can be a significant cost difference. The RCP has a fixed micro architecture that can result in fewer unused wires and routing channels compared to the much more fine-grained architectures of FPGAs. This translates to a major cost advantage for large signal processing designs mapped to the RCP.

5.3 ASIC

ASICs generally provide the ultimate in performance since they are custom designed for the application. However, ASICs fail to satisfy the market's critical time-to-market needs, and are, by definition, unable to satisfy the need for greater flexibility.

Instantaneous reconfiguration enables the RCP to compete cost-effectively with ASICs and FPGAs. For some algorithms, the data must be sequentially processed in separate processing units. The RCP can be reconfigured, thereby reusing the existing hardware for a different algorithm or application. Figure 7 shows how reconfiguration enables the re-use of hardware that might be implemented as dedicated blocks on an ASIC or FPGA.

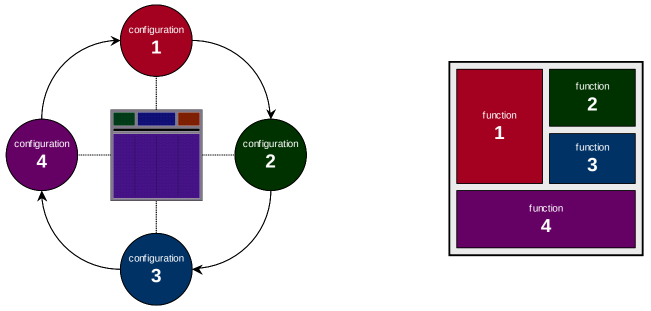

Figure 7. Reconfiguration versus Dedicated Hardware

6. Summary

Clearly, Chameleon's RCP meets the increasing computational demands of the wireless industry while providing the programmability to enable equipment providers to update deployed equipment with software programming. Validating the concept of modular computing, the RCP architecture implements an interconnected array of computational elements and memories connected with a programmable interconnect.

Hopefully, our discussion of how algorithms are designed for the RCP and where the design approach is different for conventional programmable solutions has helped to establish a clear case for reconfigurable solutions to today's processing requirements. These benefits are particular to solutions like the RCP today, but we believe that the advent of large system chips will require all applications to make these design considerations.

7. Acknowledgements

Jack Greenbaum, Kiran Bondalapati, and Mossadeq Mahmood all contributed ideas and their review to this paper.

8. References

[1] Rabey, Jan, Embedded Tutorial, ASPDAC00, Yokohama, January 2000

[2] Horowitz, Mark. Circuits and Interconnect in Aggressively Scaled MOS, *37th Design Automation Conference*, Anaheim, California, June 2000

[3] Bondalapati, Kiran, Parallelizing DSP Nested Loops on Reconfigurable Architectures using Data Context Switching, *38th Design Automation Conference*, June 18-22, 2001, Las Vegas, Nevada, USA.

[4] http://brass.cs.berkeley.edu/SCORE/

[5] http://www.ti.com/sc/docs/products/dsp/c6000/benchmarks/62x.htm