THE USE OF HARDWARE ACCELERATION IN SDR WAVEFORMS

David Lau
Altera Corporation
101 Innovation Dr
San Jose, CA 95134
(408) 544-8541
dlau@altera.com

Jarrod Blackburn
Altera Corporation
101 Innovation Dr
San Jose, CA 95134
(408) 544-7878
jblackbu@altera.com

Joel A. Seely
Altera Corporation
101 Innovation Dr
San Jose, CA 95134
(408) 544-8122
jseely@altera.com

ABSTRACT

In Software Defined Radios FPGAs can be used as both an interconnect layer and a general-purpose computational fabric implementing hardware acceleration units. Typical implementations of software defined radio (SDR) modems include a general purpose processor (GPP), digital signal processor (DSP) and field programmable gate array (FPGA). The FPGA fabric can be used to offload either the GPP or DSP (or both). Hardware acceleration units in conjunction with small embedded microprocessors can be used as coprocessors to the GPP or DSP, accelerating critical sections of either DSP or GPP code in hardware. Moreover, with general purpose routing resources available in the FPGA, hardware acceleration units can run in parallel to further enhance the total computational output of the system. The algorithms and systems can be modeled in high-level languages or tools such as C or Matlab/Simulink and easily ported to hardware acceleration units running on the FPGA. The creation and use of hardware acceleration units and their performance over software implementations will be discussed in this paper.

1. INTRODUCTION

In the past FPGAs were used as a convenient interconnect layer between chips in a system. In software defined radios (SDRs), FPGAs are being used increasingly as a general-purpose computational fabric implementing signal processing hardware that boosts performance while providing lower cost and lower power. Typical implementations of SDR modems include a general purpose processor (GPP), digital signal processor (DSP), and field programmable gate array (FPGA). However, the FPGA fabric can be used to offload the GPP or DSP with application specific hardware acceleration units. Soft-core microprocessors can have their core extended with custom logic, or separate hardware acceleration co-processors can be added to the system. Furthermore, with general purpose routing resources available in the FPGA, these hardware acceleration units can run in parallel to further enhance the total computational throughput of the system. Three distinct types of hardware acceleration units and their performance over software implementations will be discussed in this paper

2. SOFTWARE DEFINED RADIO

The concept behind SDR is that more of the waveform processing can be implemented in reprogrammable digital hardware so a single platform can be used for multiple waveforms. With the proliferation of wireless standards, future wireless devices will need to support multiple air-interfaces and modulation formats. SDR technology enables such functionality in wireless devices by using a reconfigurable hardware platform across multiple standards.

SDR is the underlying technology behind the Joint Tactical Radio System (JTRS) initiative to develop software-programmable radios that can enable seamless, real-time communication across the U.S. military services, and with coalition forces and allies. The functionality and expandability of the JTRS is built upon an open architecture framework called the software communications architecture. The JTRS terminals must support dynamic loading of any one of more than 25 specified air interfaces or waveforms that are typically more complex than those used in the civilian sector. To achieve all these requirements in a reasonable form factor requires extensive processing power of different kinds.

3. SDR SYSTEM ARCHITECTURE

Most SDR systems utilize a general purpose processor (GPP), digital signal processor (DSP), and FPGA in their architectures. The GPP, DSP, and FPGA are general purpose processing resources that can be used for different parts of the overall SDR system. Figure 1 shows the typical functions found in SDR divided across each of these devices. However there is a significant amount of overlap between each of these elements. For example, an algorithm running on the DSP could be implemented in the GPP, albeit more slowly, or rewritten in HDL and run in an FPGA as a coprocessor or hardware acceleration unit.

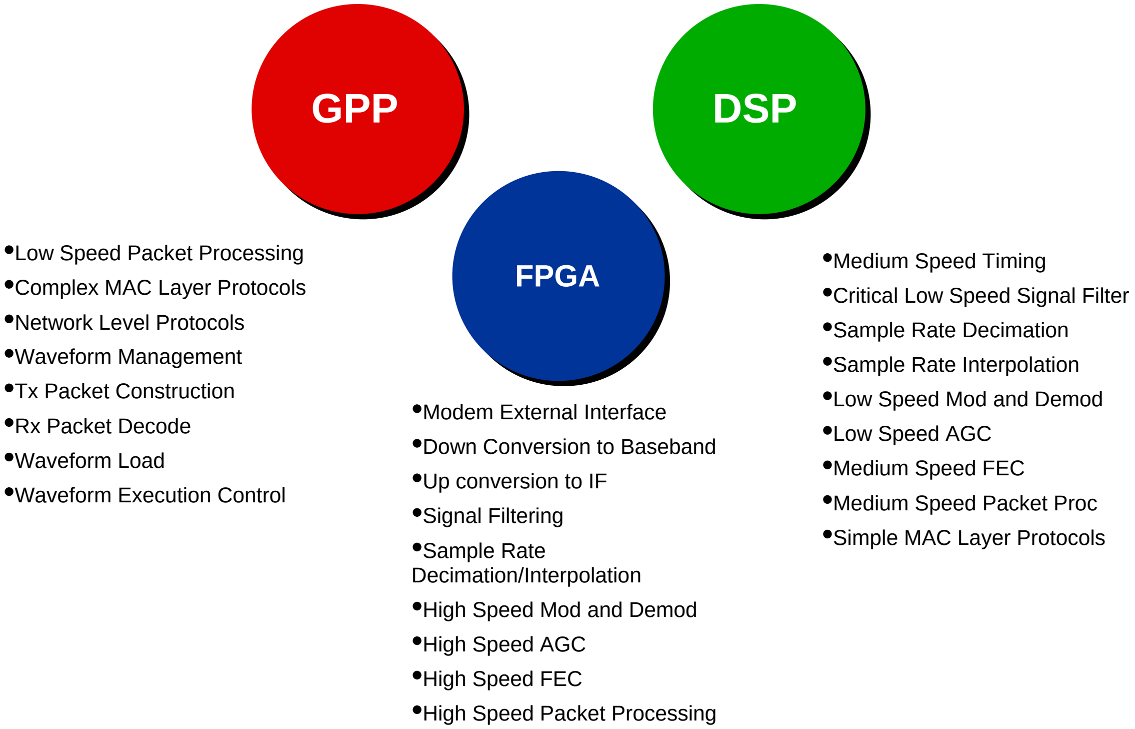

GPP
DSP
FPGA
•Low Speed Packet Processing
•Complex MAC Layer Protocols
•Network Level Protocols
•Waveform Management
•Tx Packet Construction
•Rx Packet Decode
•Waveform Load
•Waveform Execution Control
•Modem External Interface
•Down Conversion to Baseband
•Up conversion to IF
•Signal Filtering
•Sample Rate Decimation/Interpolation
•High Speed Mod and Demod
•High Speed AGC
•High Speed FEC
•High Speed Packet Processing
•Medium Speed Timing
•Critical Low Speed Signal Filter
•Sample Rate Decimation
•Sample Rate Interpolation
•Low Speed Mod and Demod
•Low Speed AGC
•Medium Speed FEC
•Medium Speed Packet Proc
•Simple MAC Layer Protocols

Figure 1: Example architecture splitting SDR functions across GPP, DSP, and FPGA

4. HARDWARE ACCELERATION

Using FPGA resources for hardware acceleration can be done in several ways. However, there are three basic architectures: Custom instructions, custom peripherals as coprocessors, and dynamically reconfigurable application-specific processors. These hardware acceleration methods have different features and unique benefits. Understanding how and where to use each of these helps the system architect better use the FPGA resources for offloading the DSP and GPP in a SDR application.

5. SOFT-CORE PROCESSORS AND CUSTOM INSTRUCTIONS

With the advent of large FPGAs, small, powerful, processors integrated in the FPGA. This is accomplished as either “hard-core” processors, which are a physical part of the FPGA silicon, or “soft-core” processors which are IP blocks downloaded as part of the design running on the FPGA. These processors, hard-core or soft-core, are used like any other embedded microprocessor. They even come with industry standard tool chains including compilers, instruction-set simulators, a full-suite of software debug tools, and an integrated development environment. This toolset is familiar to any embedded software engineer so much so that it does not matter that the processor is integrated on the FPGA or downloaded to the FPGA as a bitstream.

However, these soft-core processors have a differentiating factor over their hard-core counterparts – they are infinitely flexible. Before downloading the processor, a designer can choose different configuration options trading off size for speed. A designer can also add a myriad of peripherals, for memory control, communications, I/O, and so forth.

Custom instructions, which take the flexibility of soft-core processors one step further, are algorithm-specific additions of hardware to the soft-core microprocessor’s arithmetic logic unit (ALU). These new hardware instructions are used in place of a time-critical piece of an algorithm, recasting the software algorithm into a hardware block. A RISC microprocessor with a custom instruction blurs the division between RISC and CISC because the custom instruction units can be multi-cycle hardware blocks doing quite complex algorithms embedded in a RISC processor with “standard instructions” that take a single clock-cycle. Furthermore several custom instructions can be added to an ALU, limited only by the FPGA resources and the number of open positions in the soft-core processor’s op-code table. Figure 2 depicts the use of a custom instruction to extend the ALU of Altera’s Nios II soft-core microprocessor.

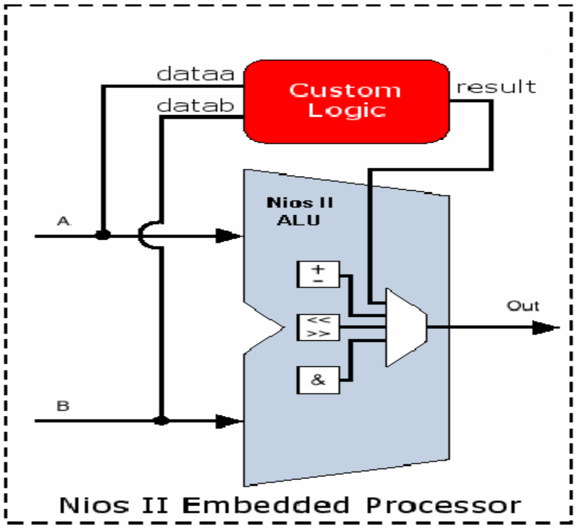


Figure 2: Diagram of soft-core processor's ALU extended with a custom instruction.

When should custom instructions be used? The most efficient use occurs when the algorithm to be accelerated is a relatively atomic operation that is called often and operates on data stored in local registers. A familiar example would be adding floating point capabilities to the processor. For many embedded processors, floating point arithmetic instructions are implemented as library subroutines that the compiler automatically invokes on processors without dedicated floating-point instruction hardware. These floating-point algorithms take many clock-cycles to execute. From an application perspective, floating point is typically used throughout the software code rather than localized to a few function calls. If the floating point instructions were part of the ALU, then these software routines would be collapsed into calls to the floating point hardware.

With soft-core processors running on an FPGA, the floating point functions can be implemented as custom instructions extending a soft-core microprocessor's ALU. The performance improvement of these hardware custom instructions over their software counterparts can be dramatic. Table 1 provides a comparison between several software library routines and the same function using a custom instruction. Note: Even in this case the results may vary depending on the design considerations for the custom instruction such as the amount of pipelining that is chosen in the hardware implementation.

The cyclic redundancy-check algorithm was also added to the table for comparison of a custom instruction in this section and a better implementation as a co-processor to be discussed in the next section. Although the CRC as a custom instruction does provide some advantages over the software only implementation of this algorithm, when the operation is executed on a large block of memory, there are other ways of implementing the hardware acceleration unit that will be more efficient and provide better overall throughput.

	CPU Clock Cycles		
Operation	Software Only	Custom Instruction	Speed Increase
FP Multiplication axb	2,874	19	151.26
FP Multiply and Negate -(axb)	3,147	19	165.63
FP Absolute |a|	1,769	18	98.28
FP Negate -(a)	284	19	14.95
CRC on 64KByte Block	2,359,312	860,179	2.74

Table 1: Comparison of Throughput Between Algorithms Implemented in Software vs. Using a Hardware Custom Instruction

6. HARDWARE ACCELERATION CO-PROCESSORS

Hardware-acceleration co-processors can be used to accelerate processors or DSPs. The processors can be either stand-alone processors separate from the FPGA, hard-core processors integrated in the FPGA, or soft-core processors downloaded to the FPGA. Custom instructions differ from hardware acceleration co-processors in that custom-instructions are an extension of an ALU which is relegated to a soft-core microprocessor. Figure 3 depicts an architecture using a co-processor. One of the key advantages to the co-processor is that it is wrapped in a DMA so it has direct access to memory – i.e. the co-processor can work on a block of memory without intervention from the processor.

Situations where hardware acceleration co-processors could be used over a custom instruction have one or more of the following common characteristics:

- Algorithms do not only use register variables (non atomic).
- Operations are more complex (often a subroutine in software).
- Transformation of data is done on a large data block.

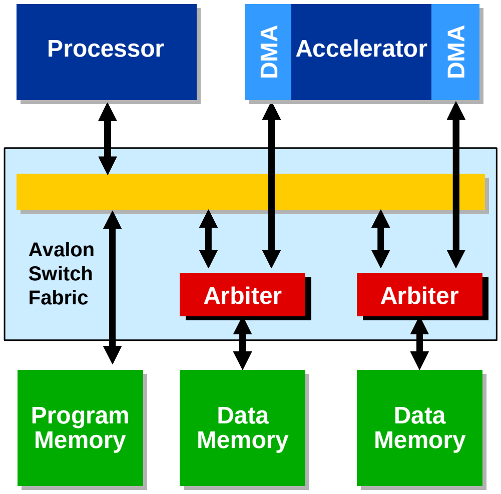


Figure 3: In-System Hardware Acceleration Co-Processor

Table 2 provides several examples of hardware acceleration co-processor performance over software only implementations for some algorithms that are used in SDR applications. SDRs can benefit from co-processors for various DSP functions as well as higher-level application level hardware acceleration.

Operation	CPU Clock Cycles		Speed Increase
	Software Only	HW Acceleration Co-Processor	
CRC on 64K Byte Block	2,359,312	43,925	53.71
Reed Solomon	25,769	985	26.16
Recursive Least Squares	13,323	1,357	9.82
FFT	179,237	11,933	15.02
Convolutional Encoder	290,858	21,869	13.30
Autocorrelation	491,030	11,955	41.07

Table 2: Software Only and Hardware Acceleration Co-Processor Throughput Comparison

7. APPLICATION SPECIFIC INSTRUCTION-SET PROCESSORS

Application specific instruction-set processors (ASIPs) are a special case of the hardware acceleration co-processors above. An ASIP combines the flexibility of a software approach with the efficiency and performance of dedicated hardware. An ASIP is a processor that has been targeted to perform a specific task or set of related tasks. One of the implementations of ASIPs allows for changing the internal topology of the processor by changing the functional interconnect of the larger building blocks. Software defined radios implement algorithms in software to improve portability, lifetime costs and retargetability. However achieving cost and performance requirements necessitates the use of application specific hardware. The value of ASIPs on an FPGA is that they are composed of smaller building blocks that can be reconfigured on the fly to implement more than one higher level function. An example relevant to SDRs would be fast Fourier transform (FFT) blocks and finite impulse response (FIR) filters. These two high-level algorithms share many common sub blocks. By changing the interconnect between these sub-blocks the ASIP can be altered to implement the FFT instead of the FIR in hardware. Figure 4 shows an ASIP architecture implementing an FFT/FIR ASIP. A simple microcode instruction set is used to configure the hardware blocks to perform either the FIR or the FFT as needed.

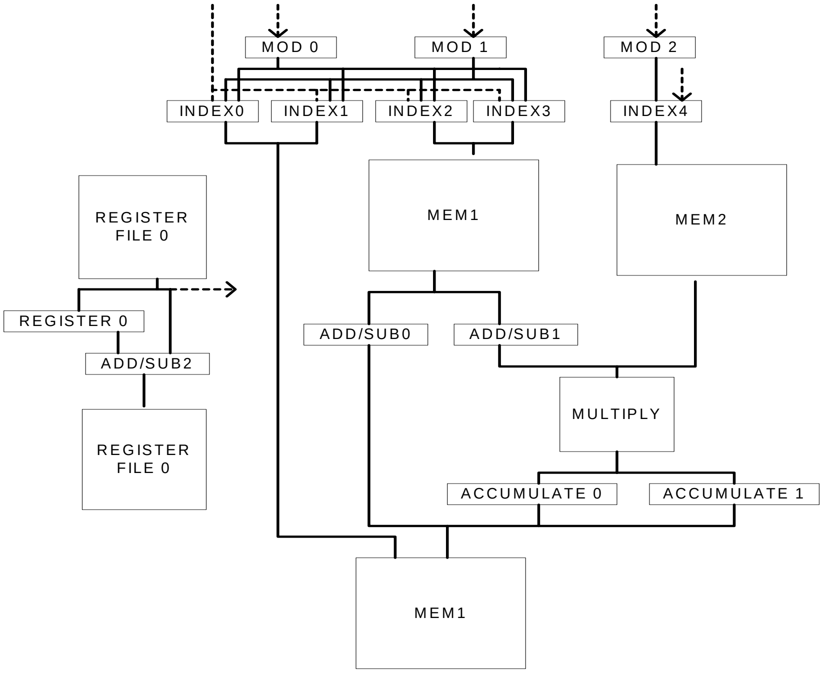


Figure 4: Combined FFT/FIR ASIP Architecture

A software/hardware comparison was made between running a 1024 point radix-2 FFT on a TI C62x DSP and doing the same filter on the FIR/FFT ASIP. The TI implementation took 20840 clock cycles and the ASIP took 21850 clock cycles. The overall throughput for both implementations was near parity. However, the relative size (and power and cost) of the ASIP approach is superior to utilization of an entire DSP for the same algorithms. In situations where specific SDR algorithms can be offloaded from the DSP to the FPGA to either decrease the processing power needed in the DSP, the results often strongly favor the ASIP approach.

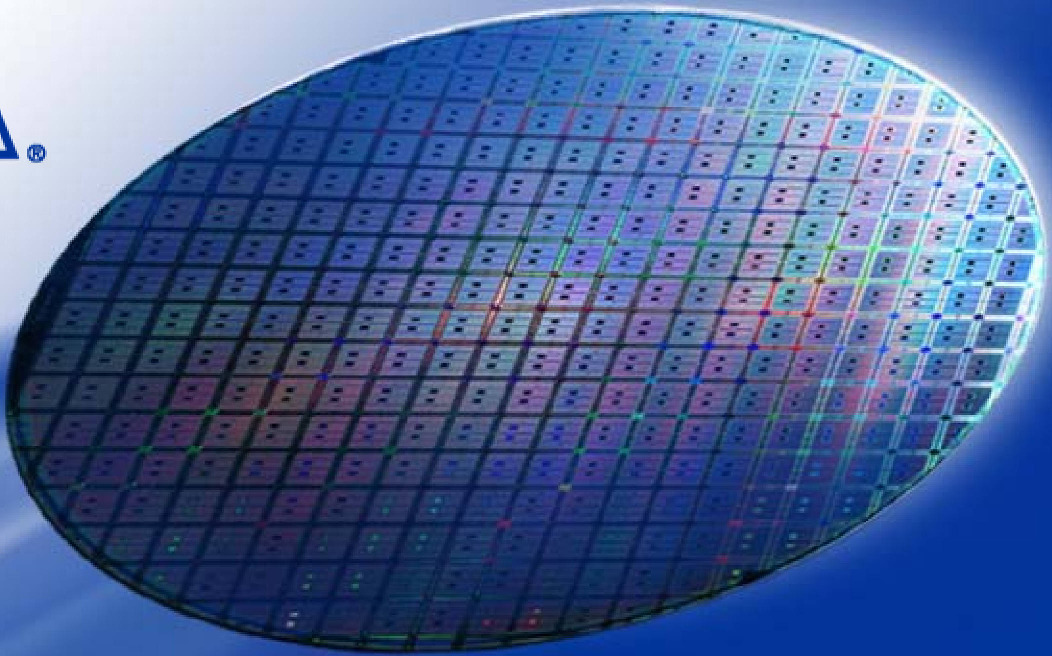
8. CONCLUSIONS

Software defined radios require extensive processing power to realize the portability of waveforms and reconfigurability that has been promised. The use of FPGAs for hardware acceleration, through custom instructions added to soft-core processors, hardware acceleration co-processors enhancing the processing power of GPPs and DSPs, or application specific instruction-set processors providing tight, efficient reconfigurable building blocks for computation, offers promising architectural options that are helping to make SDRs a reality.

9. REFERENCES

[1] Reconfigurable Radio With FPGA-Based Application-SpecificProcessors, Altera
SDR Forum Technical Conference, November 2004

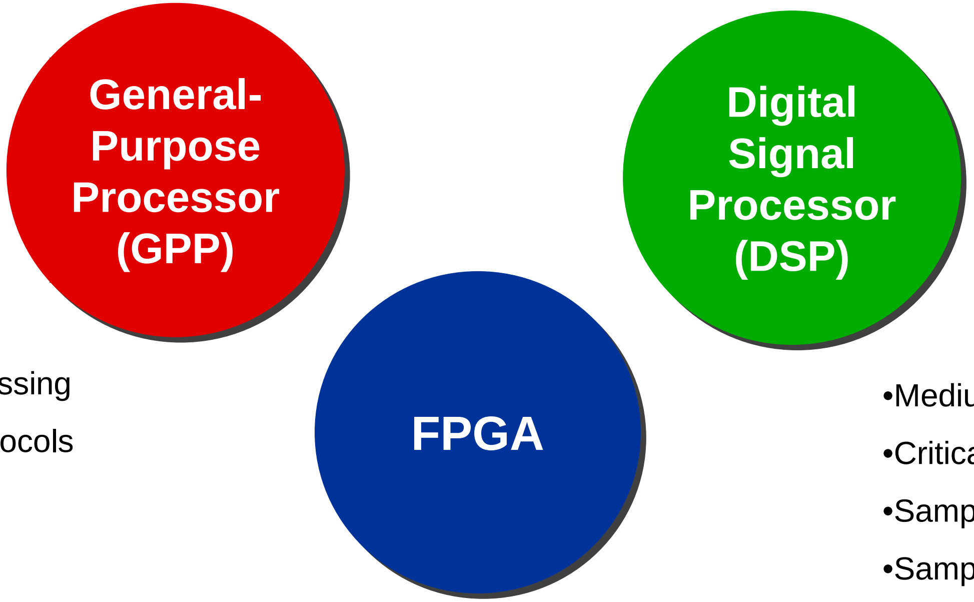
[1] Reconfigurable Antenna Processing With Matrix Decomposition Using FPGA-Based Application-Specific Integrated Processors, Altera
SDR Forum Technical Conference, November 2004



Hardware Acceleration & SDR Waveforms

Joel Seely

Waveform Processing


General-Purpose Processor (GPP)
FPGA
Digital Signal Processor (DSP)

General-Purpose Processor (GPP)

- Low-Speed Packet Processing
- Complex MAC Layer Protocols
- Network-Level Protocols
- Waveform Management
- Tx Packet Construction
- Rx Packet Decode
- Waveform Load
- Waveform Execution Control

FPGA

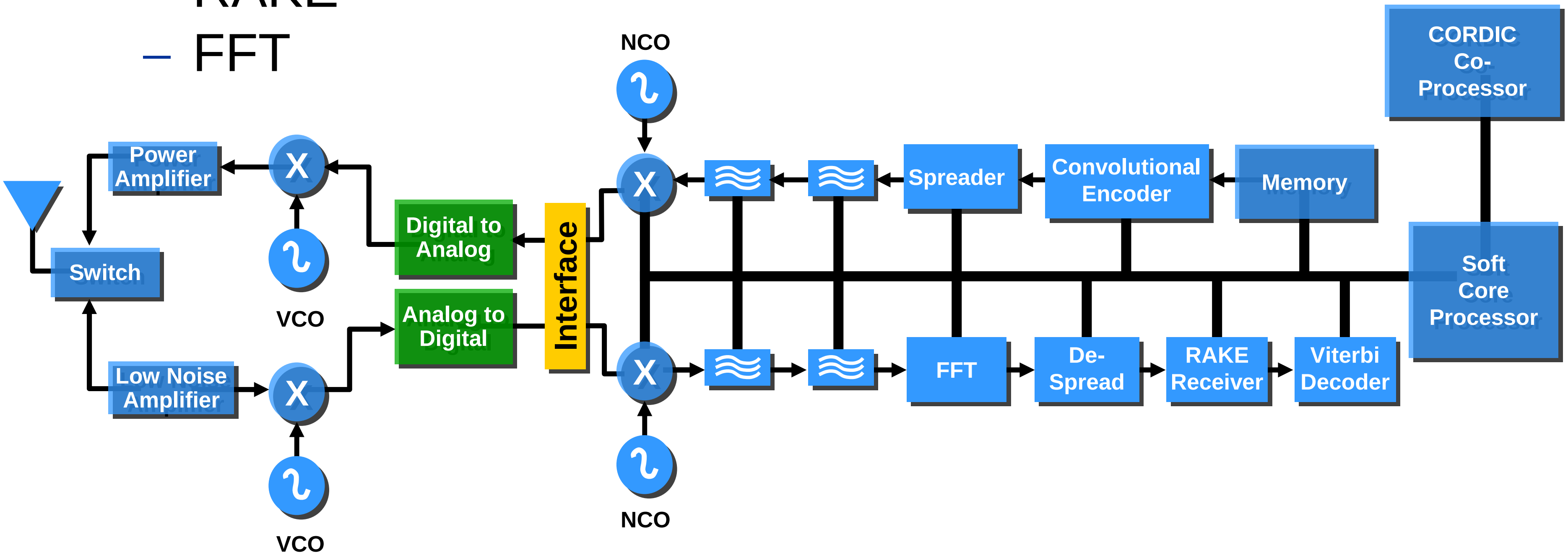
- Modem External Interface
- Down Conversion to Baseband
- Up Conversion to IF
- Signal Filtering
- Sample Rate Decimation/Interpolation
- High-Speed Mod & Demod
- High-Speed AGC
- High-Speed FEC
- High-Speed Packet Processing

Digital Signal Processor (DSP)

- Medium Speed Timing
- Critical Low-Speed Signal Filter
- Sample Rate Decimation
- Sample Rate Interpolation
- Low-Speed Modulation & Demodulation
- Low-Speed AGC
- Medium-Speed FEC
- Medium-Speed Packet Processing
- Simple MAC Layer Protocols

Waveform Example: Implementing SSW on FPGA

- Soft Processor Core for Control
- Co-Processing Engines
  - FEC
  - RAKE
  - FFT



Waveform Processing

- Different Waveforms Have Different Requirements
  - Some Are Relatively Small & Low Bandwidth
    - Can Process in DSP &/or GPP Alone
- Latest Waveforms Need High Bandwidth for Processing
  - Massively Parallel Architectures for Things like Forward Error Correction
  - Logic, Memory & MACs
  - FPGAs Required
- Waveforms Still in Development
  - SRW to be Completed in 2007

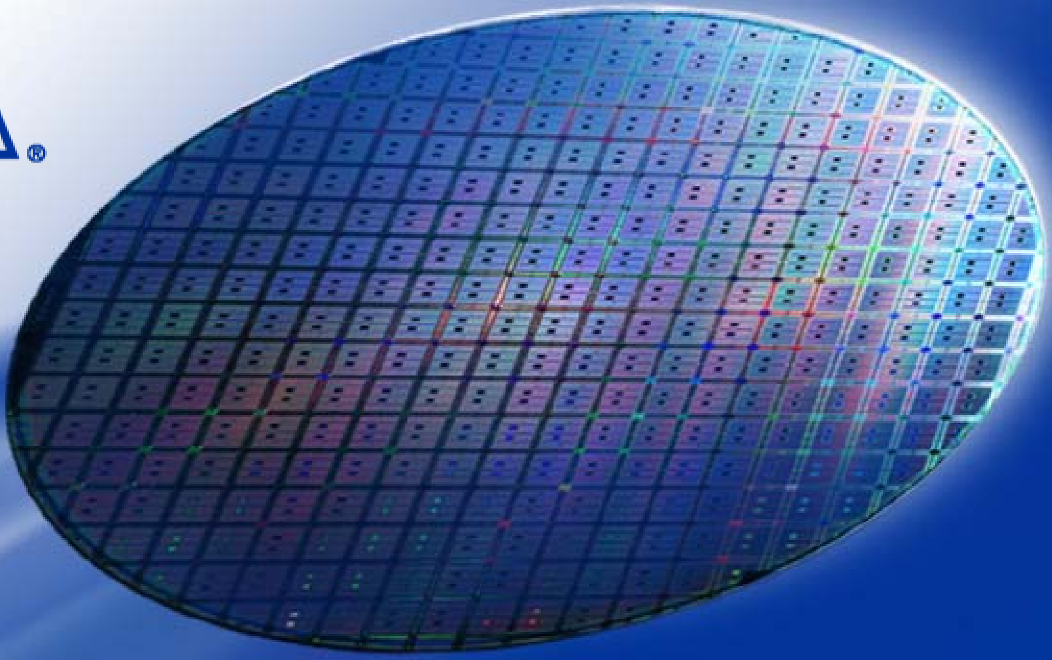
Why FPGAs for SDRs?

- Can Offload Both General Purpose Processor & Digital Signal Processor
- Glues Everything Together
- FPGAs Have Many of the Building Blocks for SDRs
  - Soft Microprocessor Cores
  - DSP Functionality (Such as Multipliers)
  - Phase-Locked Loops (PLLs)
  - Configurable I/O Pins

Hardware Acceleration

- 3 Methods
  - Custom Instructions
    - Computational
  - Co-Processor Peripherals
    - Computational
  - Application-Specific Instruction-Set Processors (ASIPs)
    - Combination Architectural & Computational
- Other Architectural Methods
  - Massively Parallel Applications
  - How You Wire Up Your Blocks Becomes Critical in High-Throughput Applications

ALTERA®



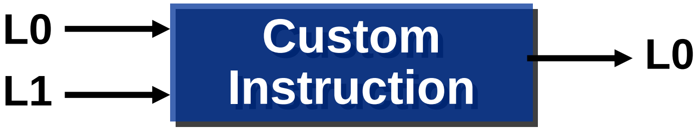
Hardware Acceleration With Custom Instructions

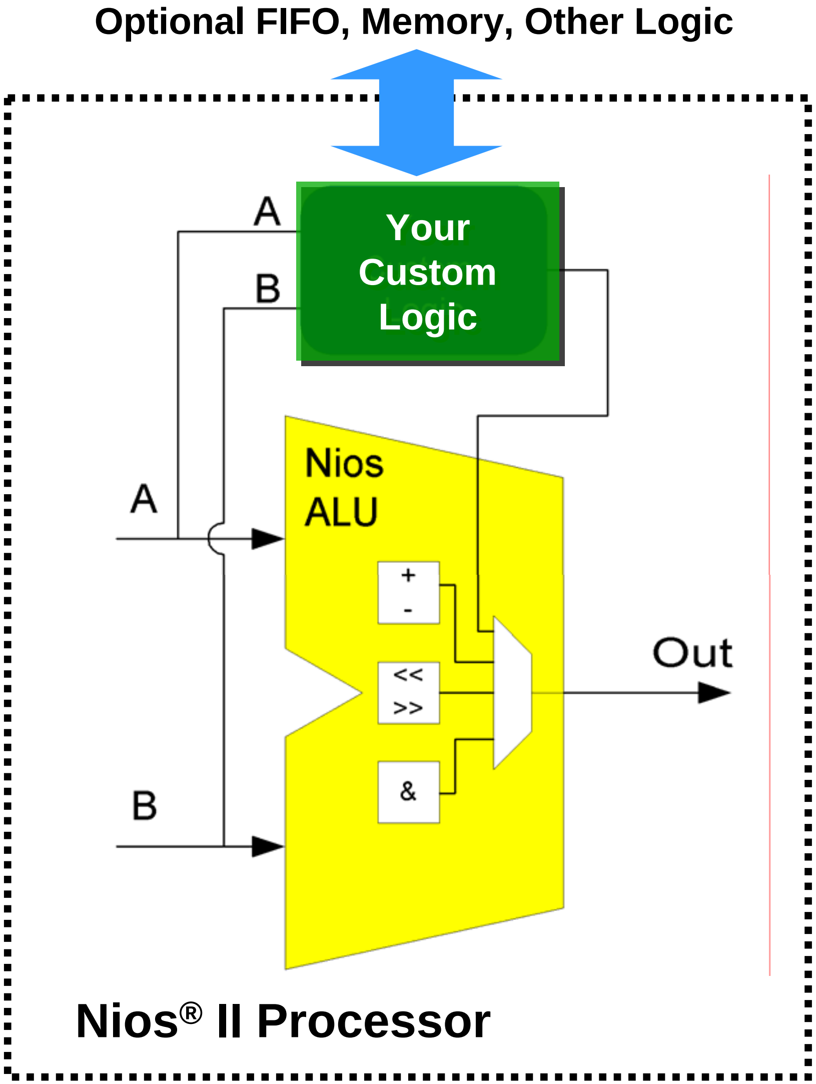
Software Bottleneck Options

- When the Processing Power of a GPP is Too Small, You Can
  - Go to Higher-Performance Processor
  - Re-Code in Assembly
  - Offload Processor to DSP
  - Offload Processor to FPGA
    - Custom Instructions
    - Co-Processors
    - ASIPs

Custom Instructions

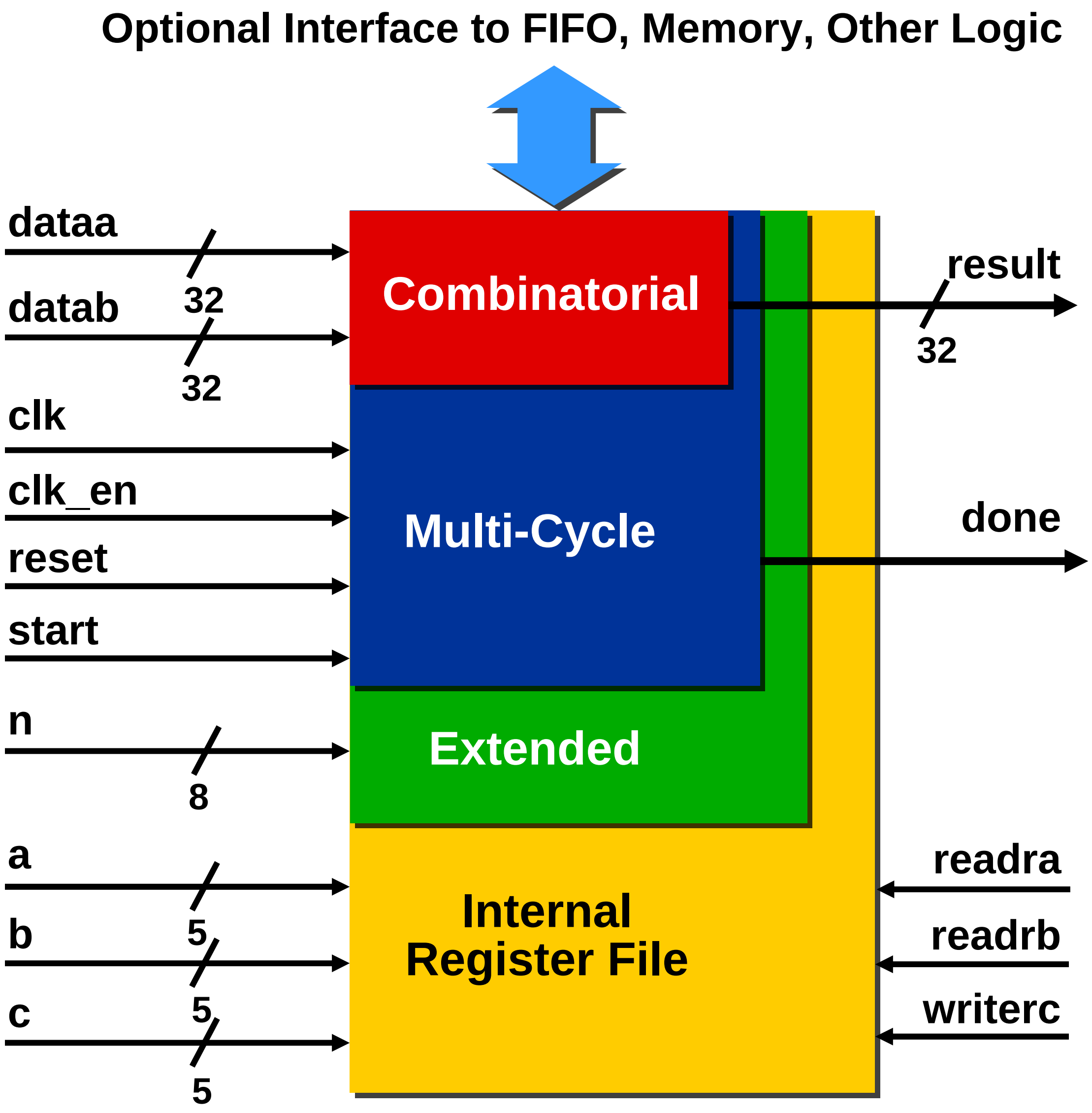
- Extend Soft-Core Microprocessors
- Can Execute in a Single Cycle
  - No Overhead for Call to Custom Hardware





© 2004 Altera Corporation - Confidential

Several Levels of Customization



Why Custom Instruction?

- Dramatically Accelerate Software Algorithms
  - Reduce Complex Sequence of Instructions to One Instruction
- Typical Flow
  - Profile Code (gprof)
  - Identify Critical Inner Loop
  - Create Custom Instruction Logic
    - Replace One or All Instructions in Inner Loop
  - Import Custom Instruction Logic Into Design
  - Call Custom Instruction From C or Assembly

When Should Custom Instructions be Used?

- Operation to be Accelerated is “Atomic”
  - Such as Math-Library Subroutines
- Multiple Clock Cycles
- Operates on Data Stored in Registers
  - Doesn’t Need to Access External Memory

Example of Custom Instruction

- Example: Floating Point Multiply

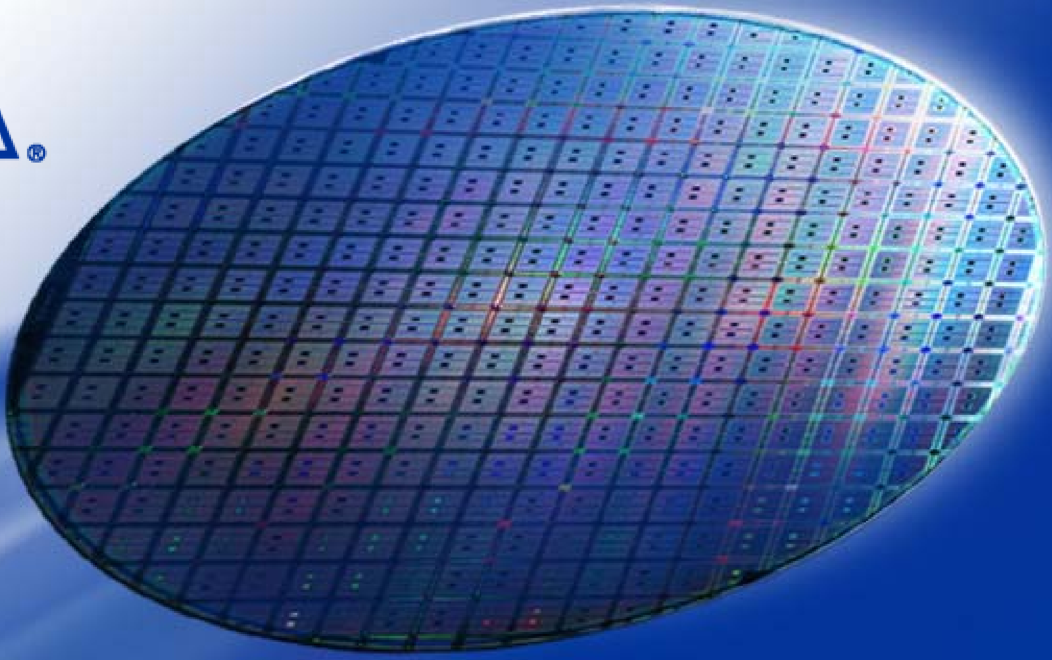
```
float a, b, result_slow, result_fast;

result_slow = a * b;        /* Takes 2,874 clock cycles  */
result_fast = nm_fpmult(a, b);  /* Takes 19 clock cycles */
```

Much Faster Than DSPs!

Custom Instruction Performance Examples

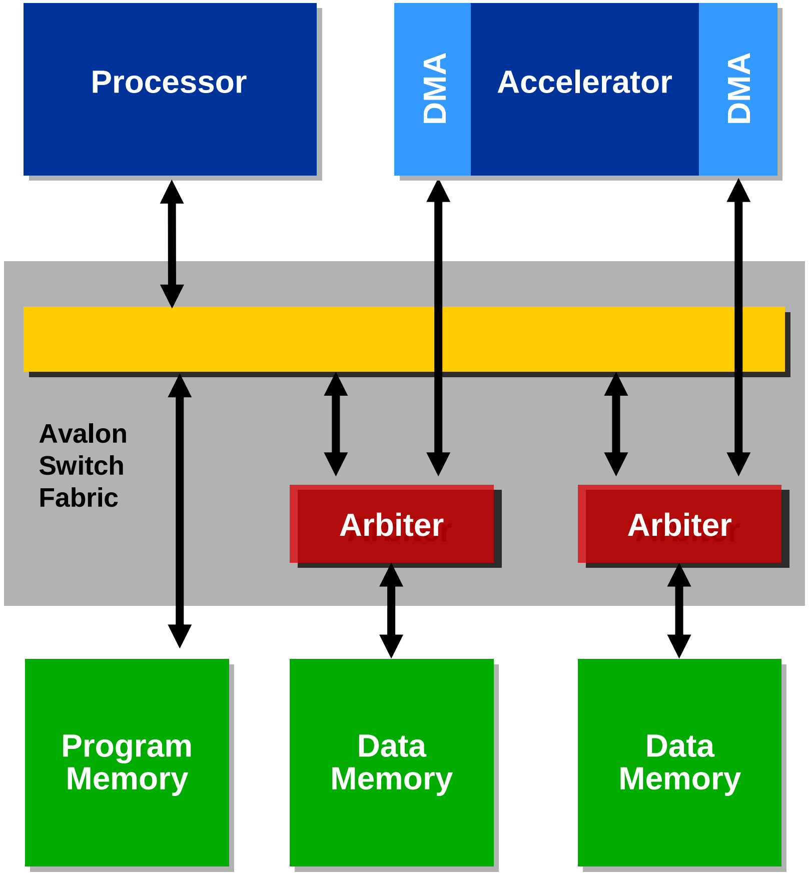
	CPU Clock Cycles		
Operation	Software Only	Custom Instruction	Speed Increase
FP Multiplication axb	2,874	19	151.26
FP Multiply & Negate -(axb)	3,147	19	165.63
FP Absolute |a|	1,769	18	98.28
FP Negate -(a)	284	19	14.95
CRC on 64-KByte Block	2,359,312	860,179	2.74



Hardware Acceleration With Co-Processor Peripherals

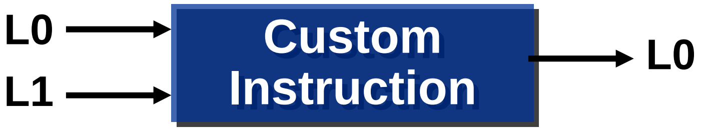
Co-Processor Peripherals

- Not Part of ALU So
  - Processor to Be Accelerated Can Be External to FPGA
  - Can Have Own Master Interface
  - Processor & Accelerator Run Concurrently

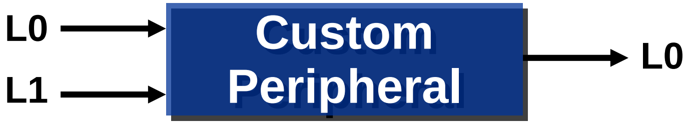


Custom Instruction vs. Co-Processor

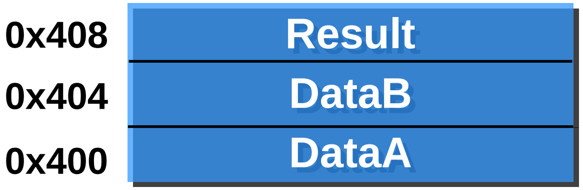
- Custom Instruction Can Execute in a Single Cycle
  - No Overhead for Call to Custom Hardware



- Access to Same Hardware as Peripheral Takes Multiple Cycles
  - Write DataA, Then Write DataB & Finally Read Result



Peripheral Memory Map



0x408	Result
0x404	DataB
0x400	DataA

  - Co-Processor Peripheral Can Run in Parallel With CPU

Co-Processor Peripherals

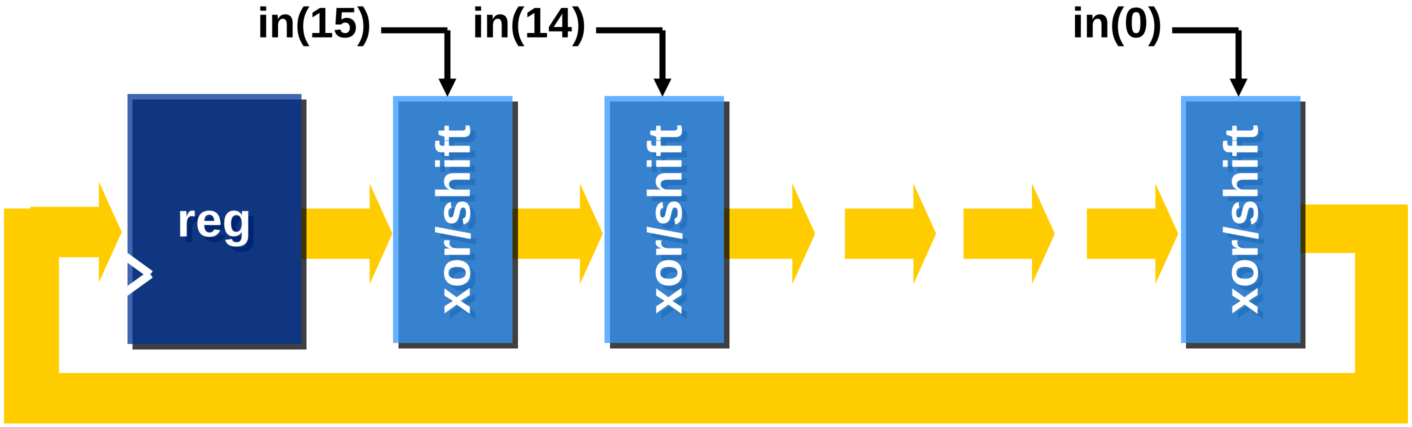
- Situations Where Hardware Acceleration Co-Processors Could be Used Over a Custom Instruction Have One or More of the Following Common Characteristics:
  - Algorithms Do Not Only Use Register Variables (Non Atomic)
  - Operations Are More Complex (Often a Subroutine in Software)
  - Transformation of Data Is Done on a Large Data Block

When Should Co-Processor Peripherals Be Used?

- Algorithms May Work on External Memory (Non Atomic)
  - Not Just Processor Registers
- Operations Are More Complex
  - Often Complex Software Subroutines
- Transformation of Data Is Done on a Large Data Block

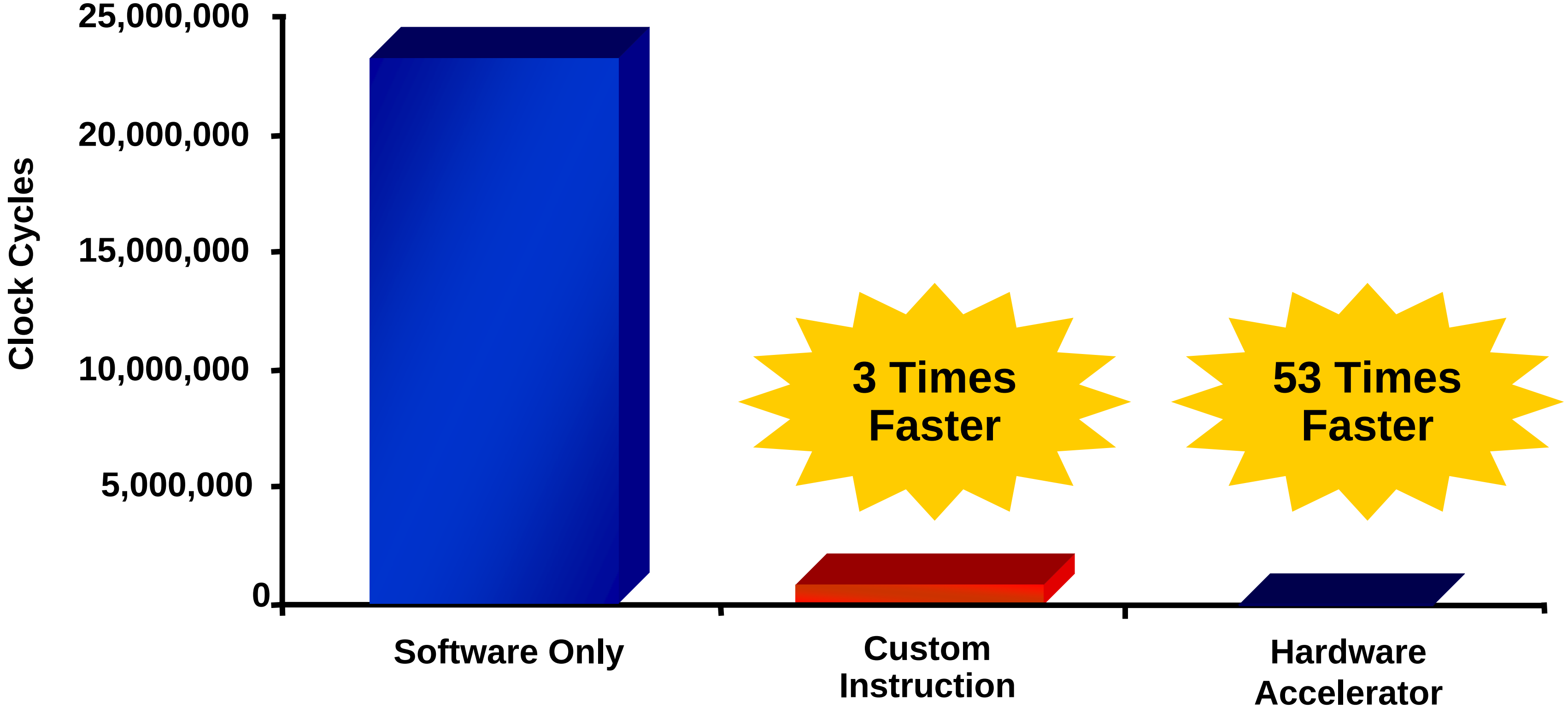
Co-Processor Peripheral Example: CRC

- Implementing the Shift & XOR for Each Bit Takes Many Clock Cycles: ~50
- Software Algorithms Tend to Use Look-Up Tables to Precompute Each Byte
- Parallel Hardware is Fastest



Accelerating Software

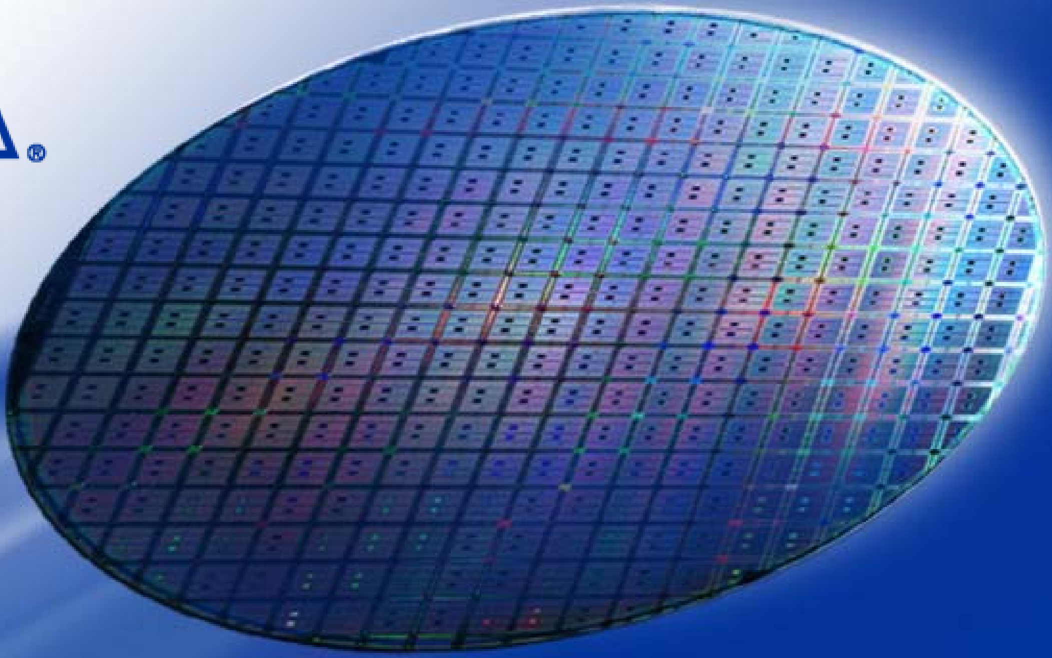
- Example: CRC Algorithm (64 Kbytes)



Co-Processor Peripheral Performance Examples

Operation	CPU Clock Cycles Software Only	 Hardware Acceleration Co-Processor	Speed Increase
CRC on 64-KByte Block	2,359,312	43,925	53.71
Reed Solomon	25,769	985	26.17
Recursive Least Squares	13,323	1,357	9.82
FFT	179,237	11,933	15.02
Convolutional Encoder	290,858	21,869	13.30
Autocorrelation	491,030	11,955	41.07

ALTERA®



Hardware Acceleration With Application-Specific Processors

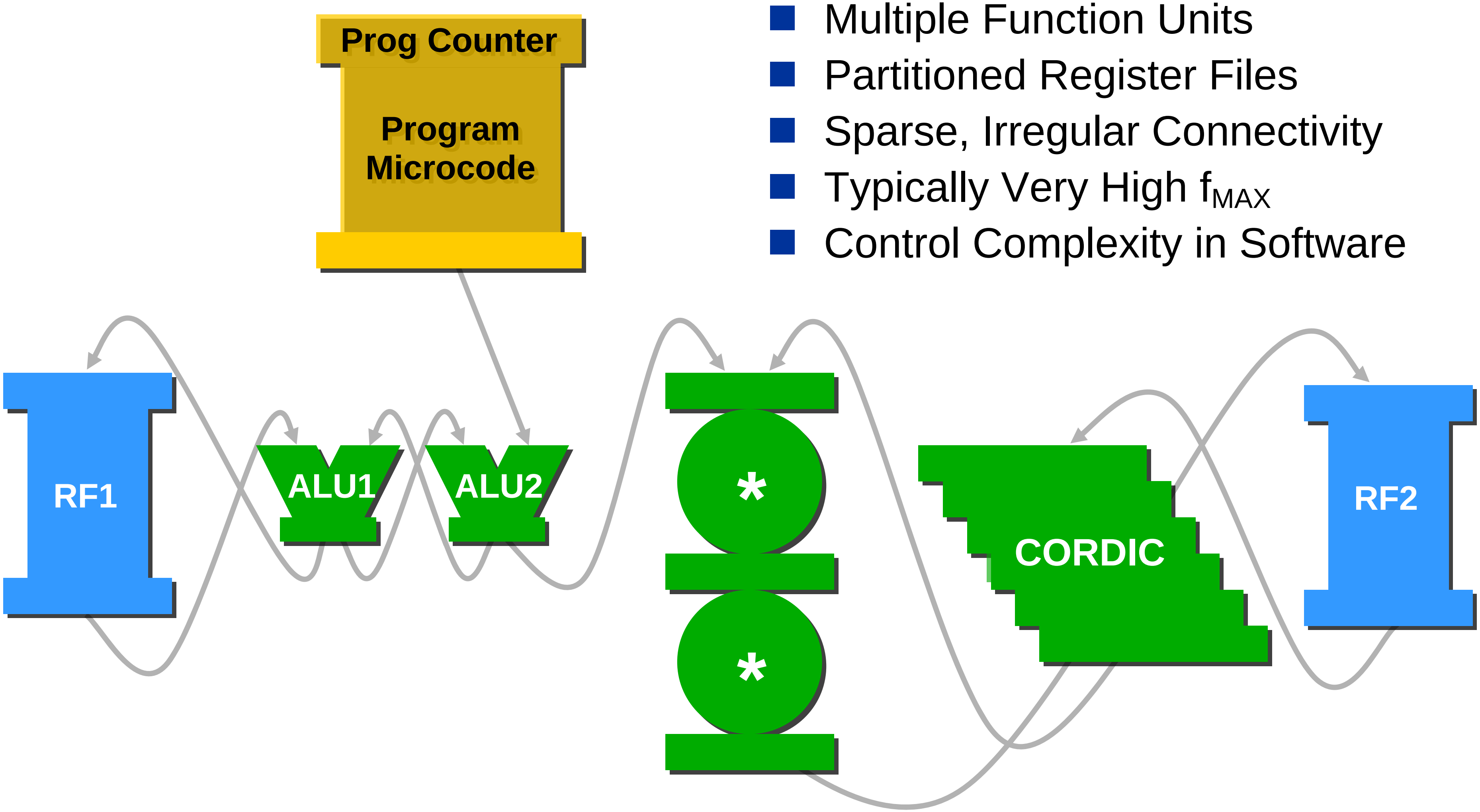
ASIP Approach

- Build Application-Specific Processors
  - Easy to Think of Them as Custom Processors
- Application-Specific Integrated Processor Has
  - Flexible Number of Functional Units
  - Connections Between Functional Units as Defined by the Algorithm
  - Program to Allow Flexible (& Dynamic) Control
  - Software-Based Design Flow

Software Process Brings Productivity vs. RTL

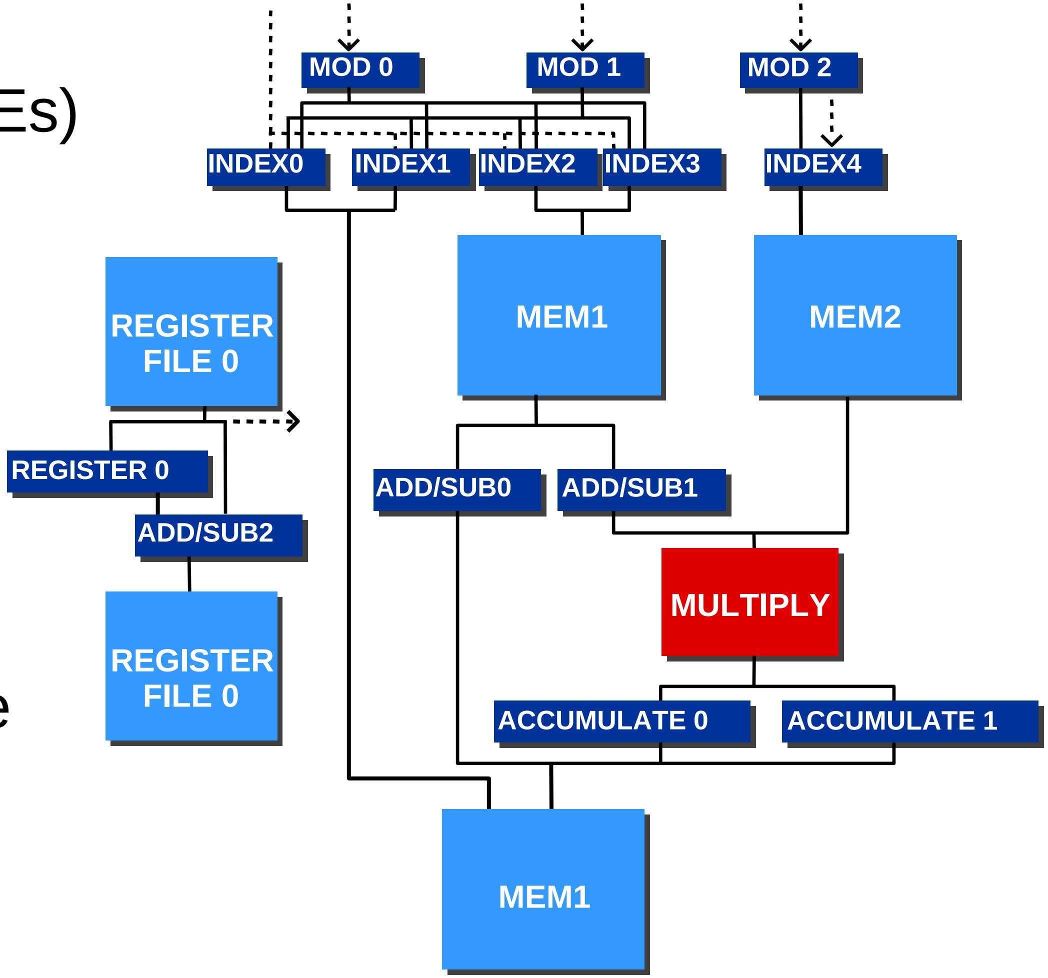
Example ASIP

- Multiple Function Units
- Partitioned Register Files
- Sparse, Irregular Connectivity
- Typically Very High f_{MAX}
- Control Complexity in Software



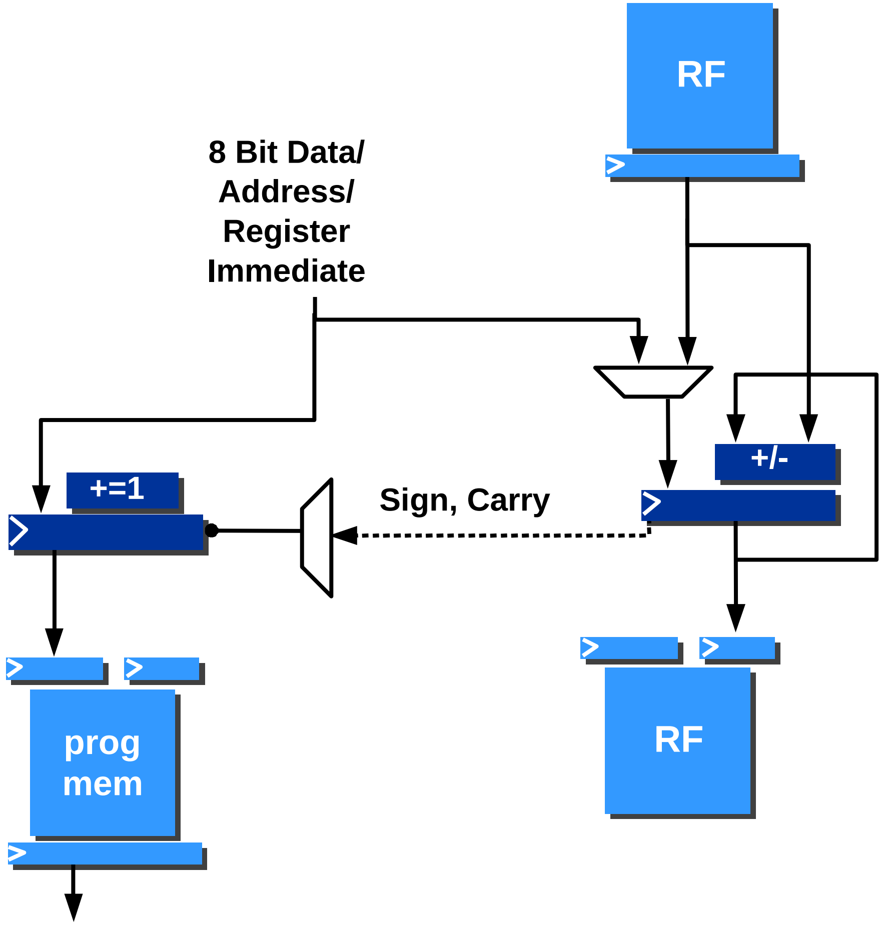
Reconfigurable FFT/FIR Filter Module

- Simple Hardware
  - 322 Logic Elements (LEs)
  - 1 18x18 Multiplier
- High Performance
  - 230 MHz in a Stratix –5 FPGA
  - 1,024 Point FFT Takes 21,850 Cycles (95 µs)
- Less than 3% of Available Logic in Smallest Altera® Stratix II FPGA



FFT/FIR Control Processor

- Complete Processor
- Can Perform Any Task
- 52 Logic Elements (LEs) & 2 M4K Memories
- Processors Implement Control Structures Very Cheaply & Densely



Reconfiguration & ASIPs

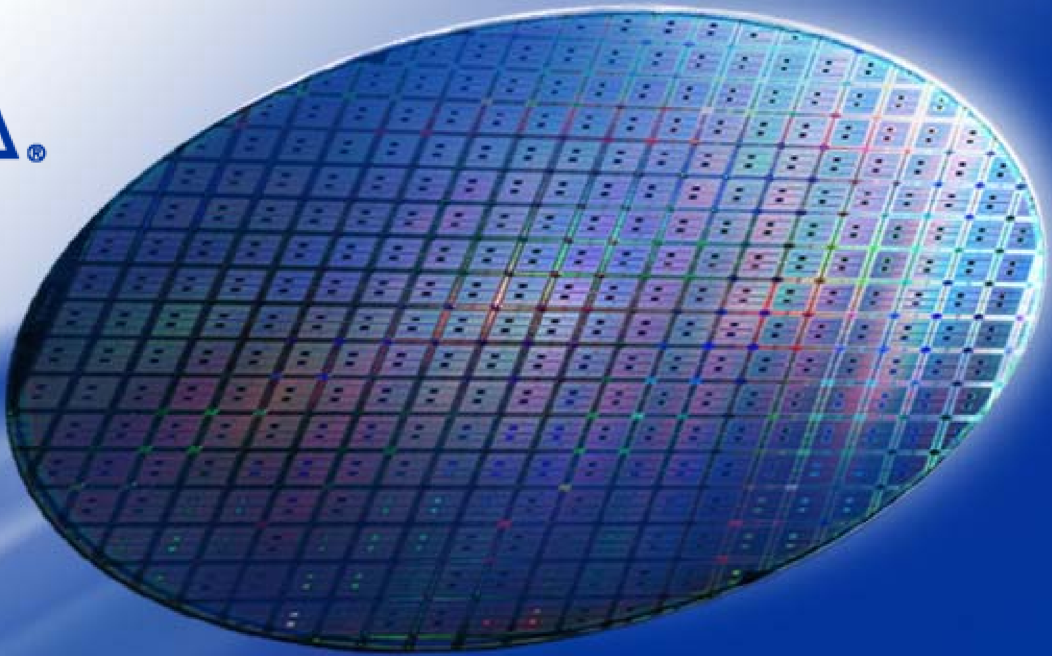
- ASIPs Use a Program
  - Sequential Instruction-Level Parallelism
  - Can be Quickly & Easily Reprogrammed by Changing the Contents of Program Memory
- Conventional Hardware Description
  - Behavior Encoded in a Number of Parallel Processes
  - Reconfiguration Requires New FPGA Image - Literally Rewiring the Device
- ASIPs Can Combine Flexibility of DSPs with Processing Power of FPGAs
- Migrate to Structured ASIC (e.g., Altera HardCopy® Structured ASIC)
  - Does Not Compromise the Reconfigurability

Benchmark Results

Application	Technology	Size (LEs)	f_{MAX} (MHz)	18x18 Multi-pliers	Actual (Clocks)
Block-FIR 40 samples, 16-tap 16-bit data	TI C54		160		793
	ASIP	56	278	1	640
IIR 2-Biquad	TI C54		160		17
	ASIP	74	278	1	8
FFT (1,024 pt, 16-bit)	TI C54		160		42,000
	ASIP (Radix2)	332	221	1	21,000
		302	236	4	11,092
	ASIP (Radix4)	578	229	4	5,335

Summary

- FPGAs Provide Several General Methods for Hardware Acceleration
  - Custom Instructions
  - Co-Processor Peripherals
  - ASIPs
- Performance Improvements Come Through
  - Accelerating Algorithm
  - Architectural Enhancements
- These FPGA Hardware Acceleration Methods are Complimentary to GPP & DSP in SDR Systems



Thank You

Joel Seely