Floating Point Functional Cores For Reconfigurable Computing Systems

By
Clay Gloster. Jr., Ph.D., P.E.

Department of Electrical & Computer Engineering,
Howard University

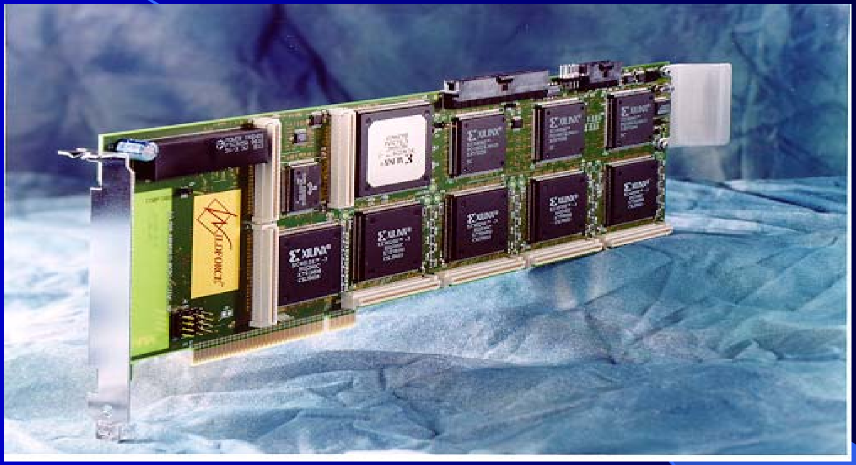
Supported under NASA Contract # AIST-0016-0044
June 25, 2003

Hierarchical Algorithms and their Embedded Computational Realization in Reconfigurable Hardware

PI: Clay Gloster/Howard University
Proposal No: AIST 0016-0044

escription and Objectives

project addresses problems associated with developing products for deployment in onboard RC systems. It ves the development of a compiler that reads algorithm riptions written in C. The compiler will produce ware and software components required for an RC ementation of typical NASA data products. The main ctives of this project are: efficient algorithm lopment and fast and reconfigurable hardware ementations (10X-100X speedup).



pproach

elop a compiler to translate nested loops into a ence of floating point vector instructions. These uctions correspond to modules in a library that is to eveloped as a part of this project. Hardware ules will perform complex instructions i.e. nult, vec-vecmult, FFT, LU Decomposition, etc.

o-PIs/Partners

nid Krim, Tom Conte, NC State University

mas Flatley, NASA GSFC

Schedule and Deliverables

- Prototype RC Test bed shown above (10/02)

-Prototype Compiler (10/02)

-Application Demonstration (10/03)

-Final Compiler (10/03)

Application/Mission

Data Product Development for EOS/AM-1 Satellite

Outline of this Presentation

- Background: Floating Point Numbers
- Introduction to Reconfigurable Computing
- A Compilation Tool for Reconfigurable Computing Systems
- A Reconfigurable Processor
- Floating Point Functional Cores
- Functional Core Performance / CLB Utilization

Floating Point Numbers

- Floating point number representations allow us to use real numbers on a computer
- Floating point numbers consist of a sign, mantissa, base, and exponent
  - $+10.34 \times 10^{32}$
  - $+1.034 \times 10^{33}$
- Since each floating point number can be represented an infinite number of ways, we normalize the number.
  - $+1.034 \times 10^{33}$

Floating Point Numbers
IEEE Single Precision

1 bit	8 bits	23 bits
sign	exponent	mantissa

Total 32 bits

- Most computers support single (32-bit) precision formats

- Single precision format can express numbers from (-3.4 E 38 to 3.4 E 38)

Floating Point Addition (Complex)

To add two floating point numbers we:

- **Align exponents while adjusting the mantissa of one operand**
- **Add resulting mantissas**
- **Compute the sign of the result based on the sign and magnitude of the two operands**
- **Normalize the result**

Floating Point Addition

0.5 + 0.5 = ????
A + B = Q

A = 0.500000
A = 3f000000
Sign A: = 0
Exponent A: = 01111110
Mantissa A: = 00000000000000000000000
Real Exponent = -1
Mantissa = 0

B = 0.500000
B = 3f000000
Sign B: = 0
Exponent B: = 01111110
Mantissa B: = 00000000000000000000000
Real Exponent = -1
Mantissa = 0

Floating Point Addition

0.5 + 0.5 = 1.0
A + B = Q

Q = 1.000000
Q = 3f800000
Sign Q: = 0
Exponent Q: = 01111111
Mantissa Q: = 00000000000000000000000
Real Exponent = 0
Mantissa = 0

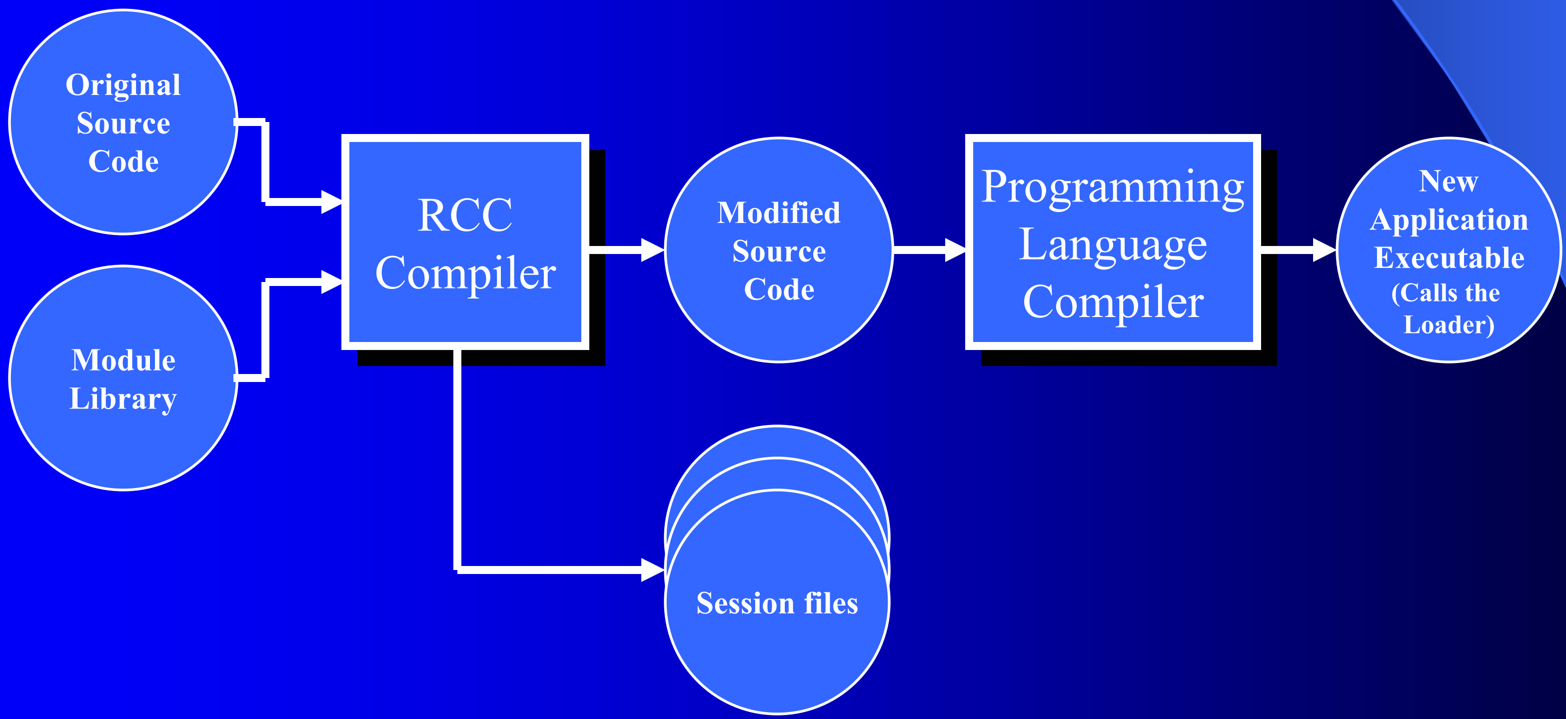
Reconfigurable Computing

- Field Programmable Gate Arrays (FPGAs) are hardware reprogrammable integrated circuits that consist of an array of programmable gates, flip-flops, programmable pins, and a programmable interconnection network.
- One function can be loaded into an FPGA now (i.e. matrix multiplication) and another function (i.e. LU decomposition) can be loaded into the FPGA later.
- FPGAs have been used as coprocessors to typical processors (forming a reconfigurable computer) to speedup several applications by orders of magnitude.
- However, users of reconfigurable computers must be knowledgeable in both hardware design as well as software development.
- Additionally, mapping an application to a reconfigurable computer is tedious and can be time consuming.

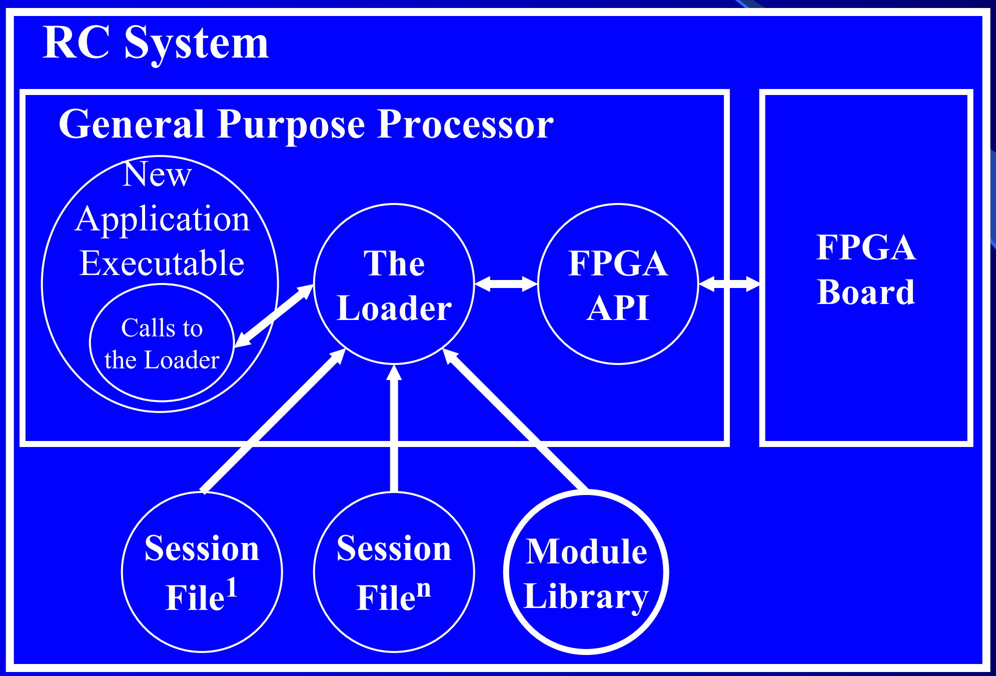
The RCC Compiler

- The purpose of the compiler is to map user applications to FPGA-based reconfigurable computers (RC), (i.e. the BISON reconfigurable computer).

- The compiler takes the original source code written in C/C++ and a module library and produces two outputs: the modified source code and a session file for each modified section.



The Execution Phase: Running the Application on the RC



The Module Library

- A hardware module is a pre-compiled, placed and routed, configuration file that is to be loaded into a specific FPGA device.
- It is a **configurable instruction set microprocessor** with a small number of instructions (Load, Store, Halt, …, CoreOp).
- The module library includes several hardware modules that have been described to the compiler.
- A module is used during the execution phase and executes operations found in nested loops in the original program that are the bottlenecks of CPU execution.

Module: Matrix Multiplication

Instructions:
Load, Store, Halt,
CpyMem, and MatMult

Hardware Modules

- All modules were designed to perform 32-bit floating-point (FP) operations.
- Standard components were developed to reduce the development time of the modules for new FPGA boards.
  - Standard functional core units. (Types I and II)
  - Standard Datapaths.
  - Standard Controllers.
- By combining standard components, several modules were developed to cover a wide variety of vector operations.
- Modules were implemented in VHDL. However, we are currently using VHDL generators to generate standard controllers and function cores.

The Benefits of Floating Point Modules

- Applications requiring many significant digits of precision or a large dynamic range can be developed simply.

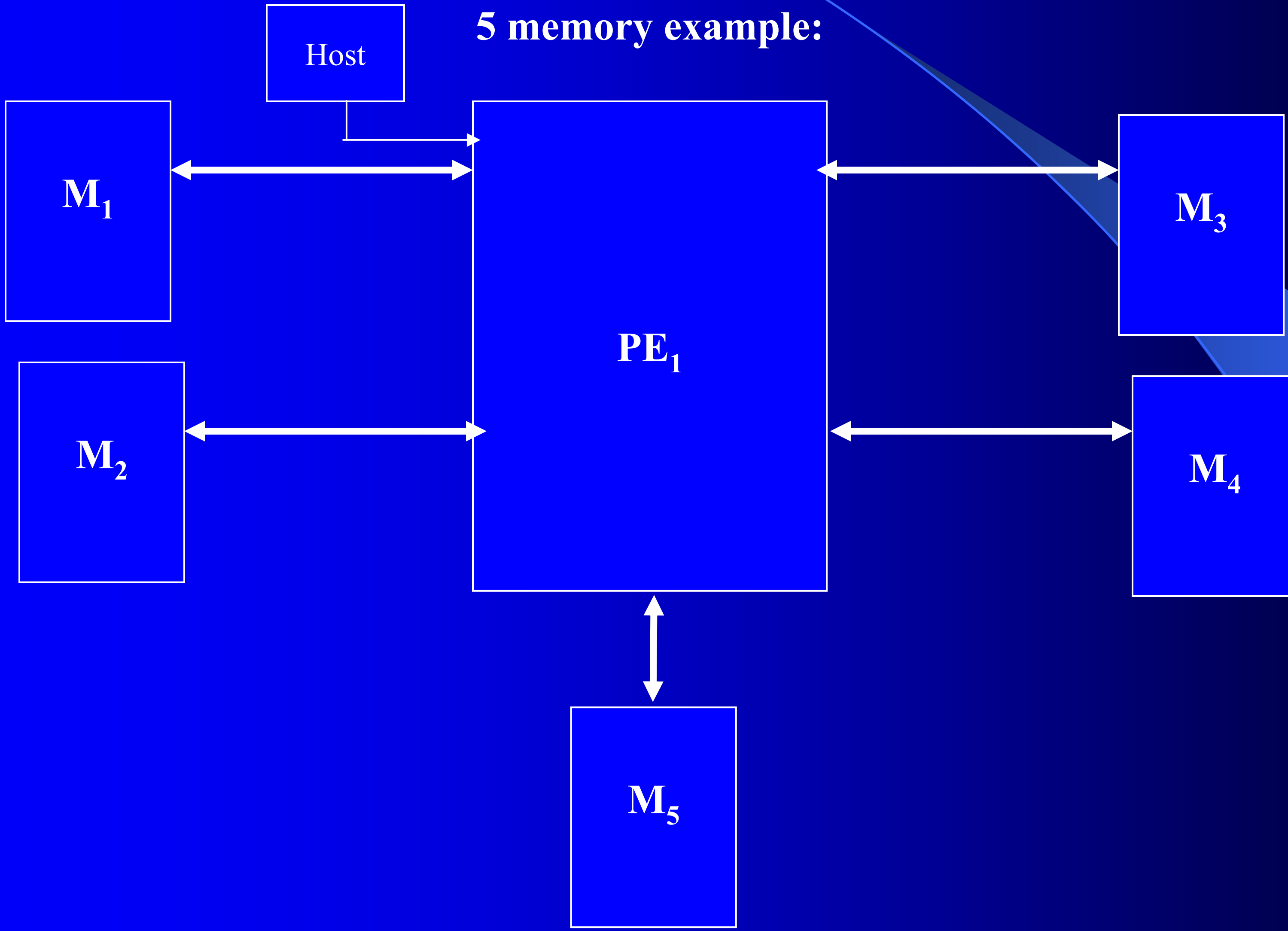
- Hardware debugging is simplified if the user was given an application previously written using a programming language that contained floating point arithmetic operations.

- There is no need for precision analysis and conversions between fixed point and floating point numbers.

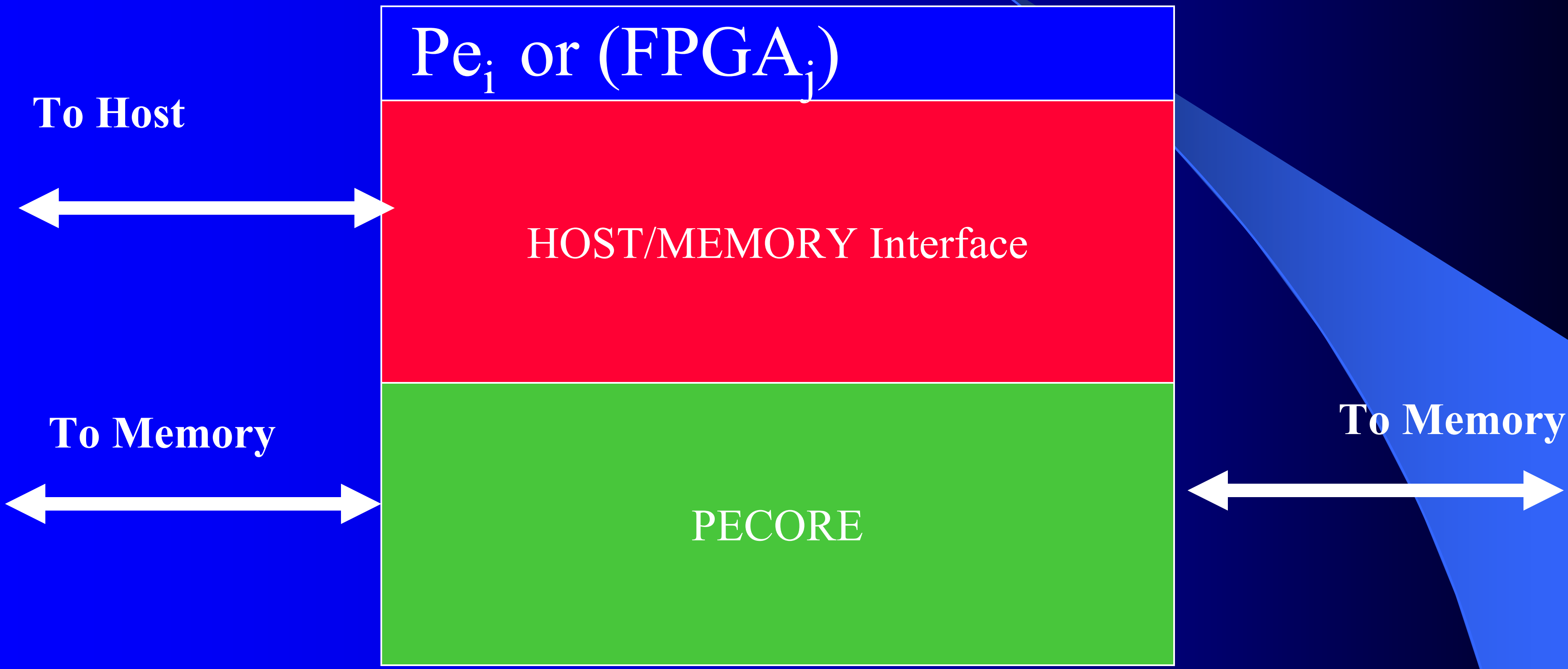
- Floating point RC systems can provide significant speedups and can fit into current FPGA-based systems.

A Reconfigurable Processor

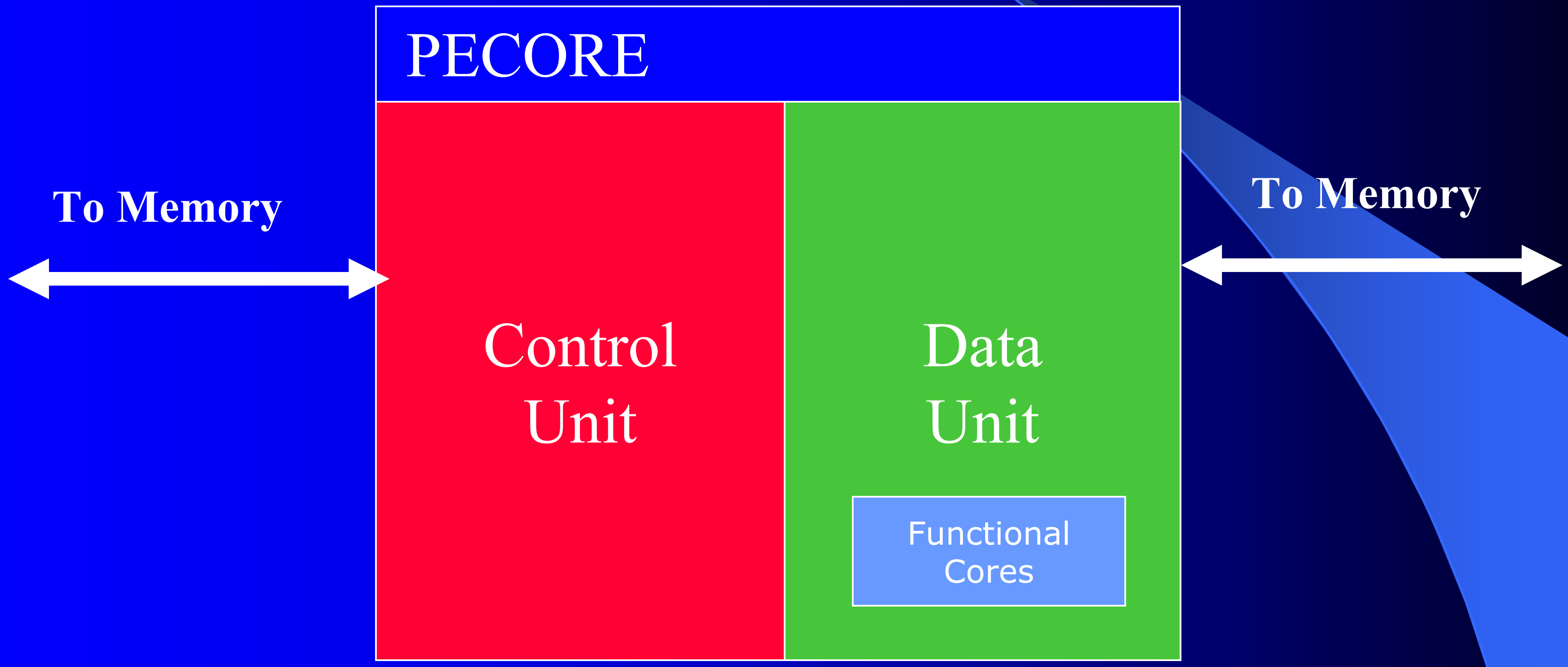
5 memory example:



The Processing Element Architecture



The PE Core


PECORE
To Memory
Control Unit
Data Unit
Functional Cores
To Memory

The Control Unit

- Manages memory read/write transactions.

- Initiates instruction fetch/decode/execution

- Determines when instruction processing is complete and turns control back over to the Host/Memory Interface.

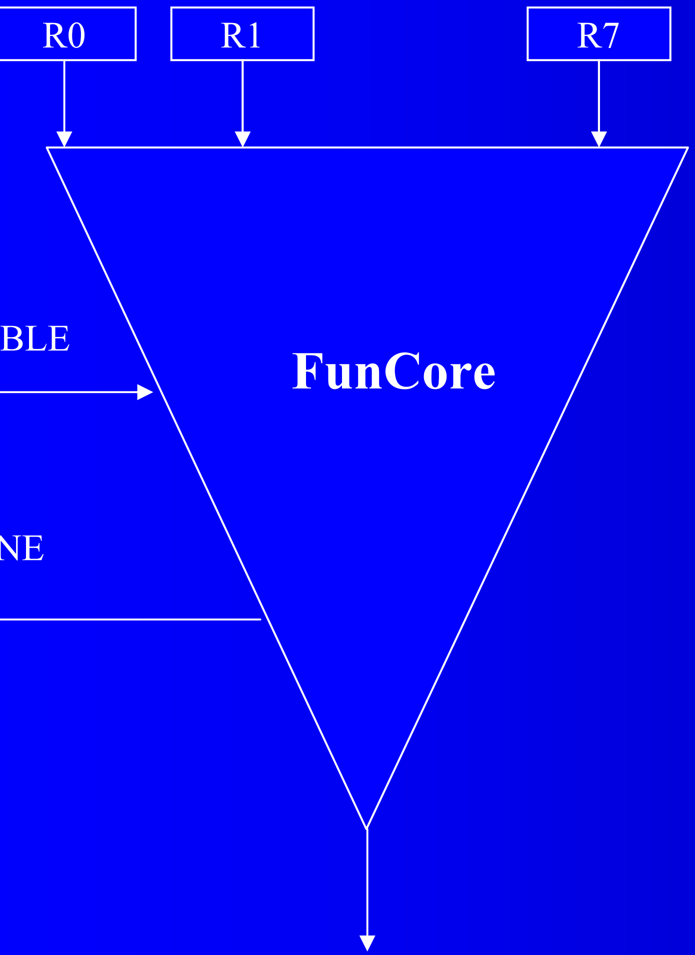
- One controller handles processing for all hardware modules/instructions.

- Changes to the controller are made for each new FPGA board based on the vendor’s supplied memory interface.

The Data Unit

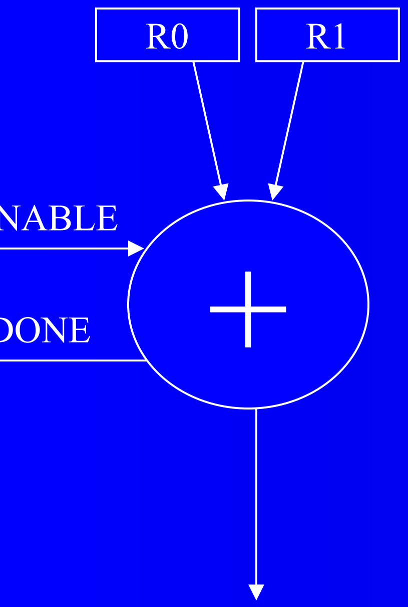
- Contains a register file (8 32-bit registers) and counters for determining when vector instructions are complete.
- Contains several memory address registers/counters for indexing through input/output vectors.
- Contains up to 7 Functional Cores (FunCores).

Functional Core Definition (Type I)

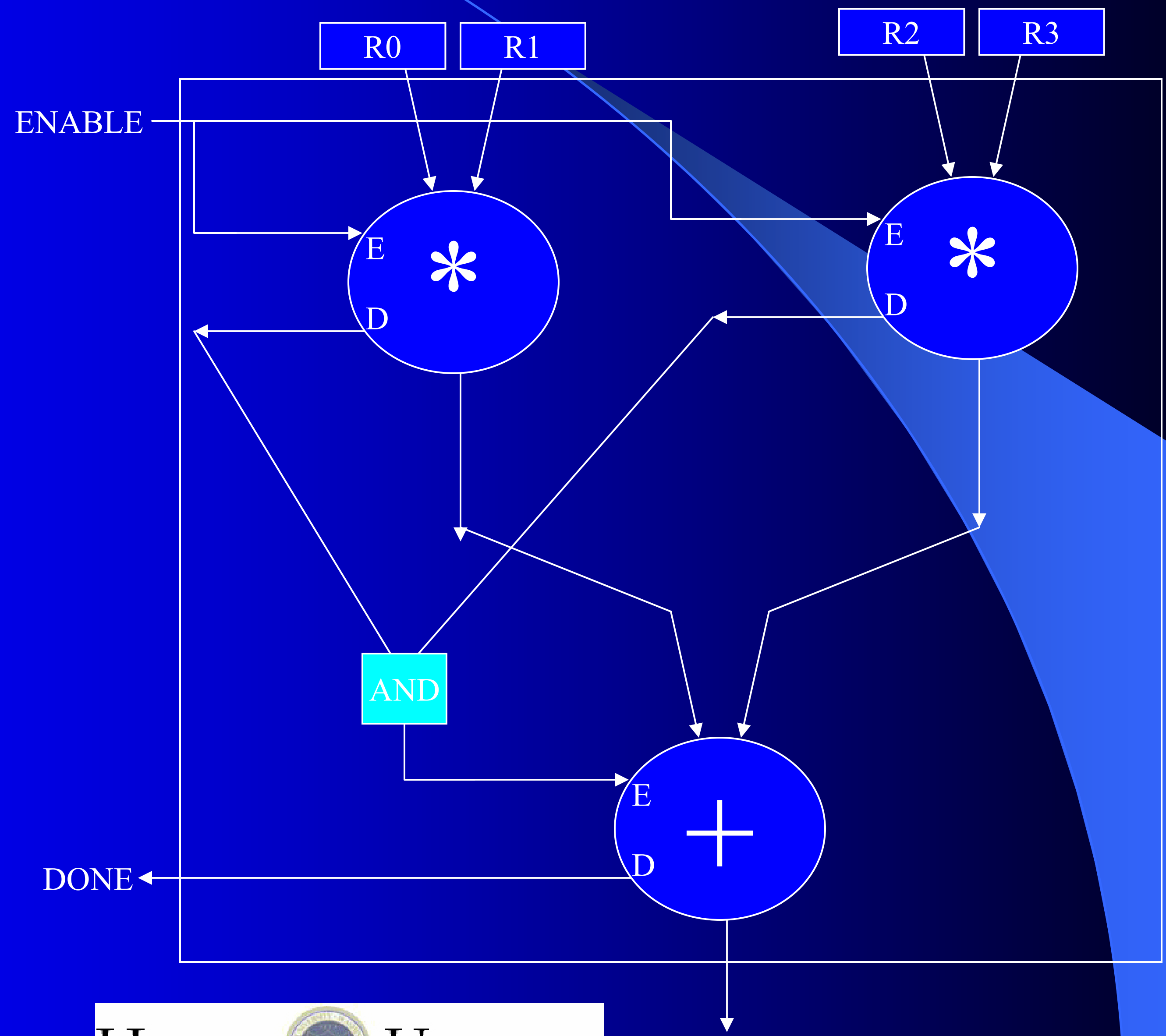


- Has one or more 32-bit inputs
- Performs floating point vector operations.
- Has simple control.
- Can be built using other FunCores.
- Can include conditional units.

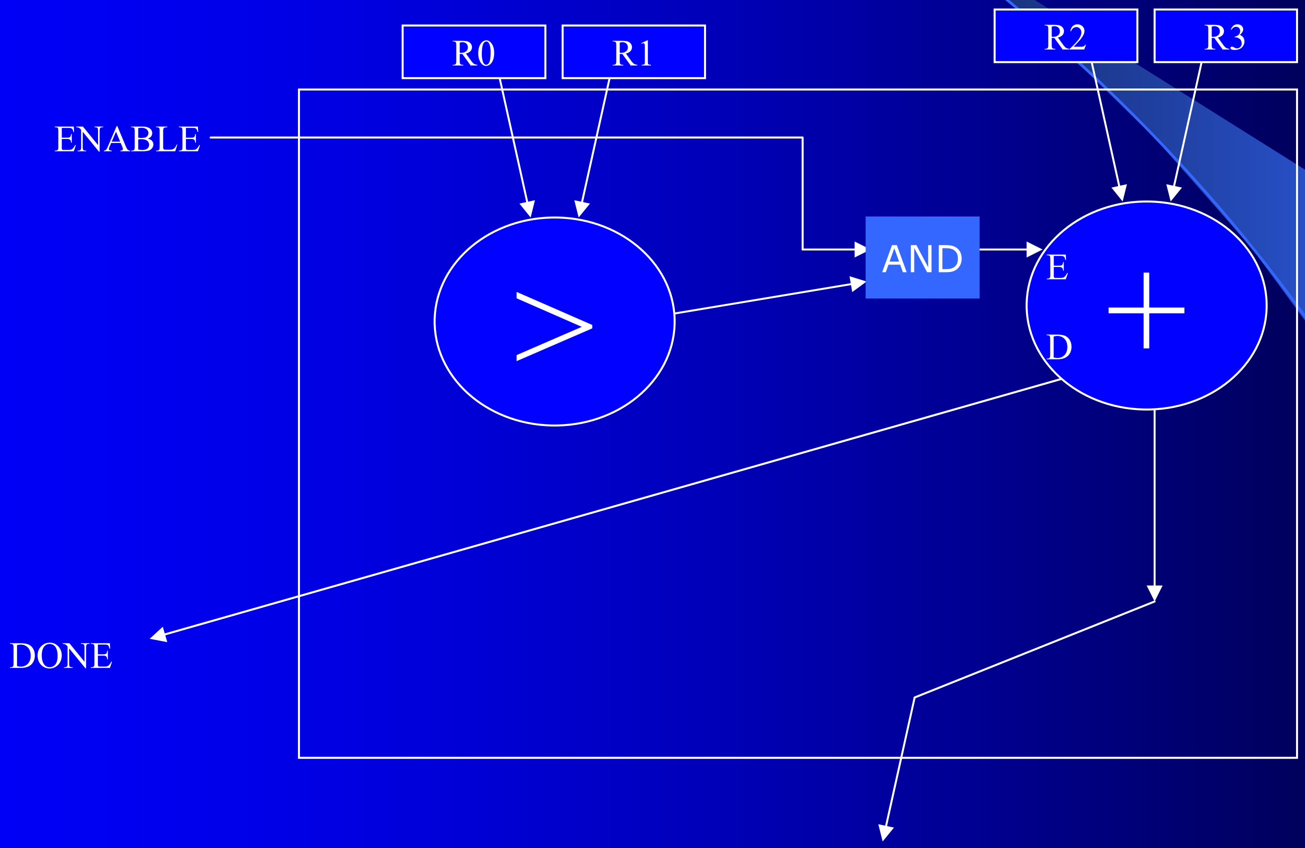
Sample Functional Cores (Type I)



Q = R0 + R1

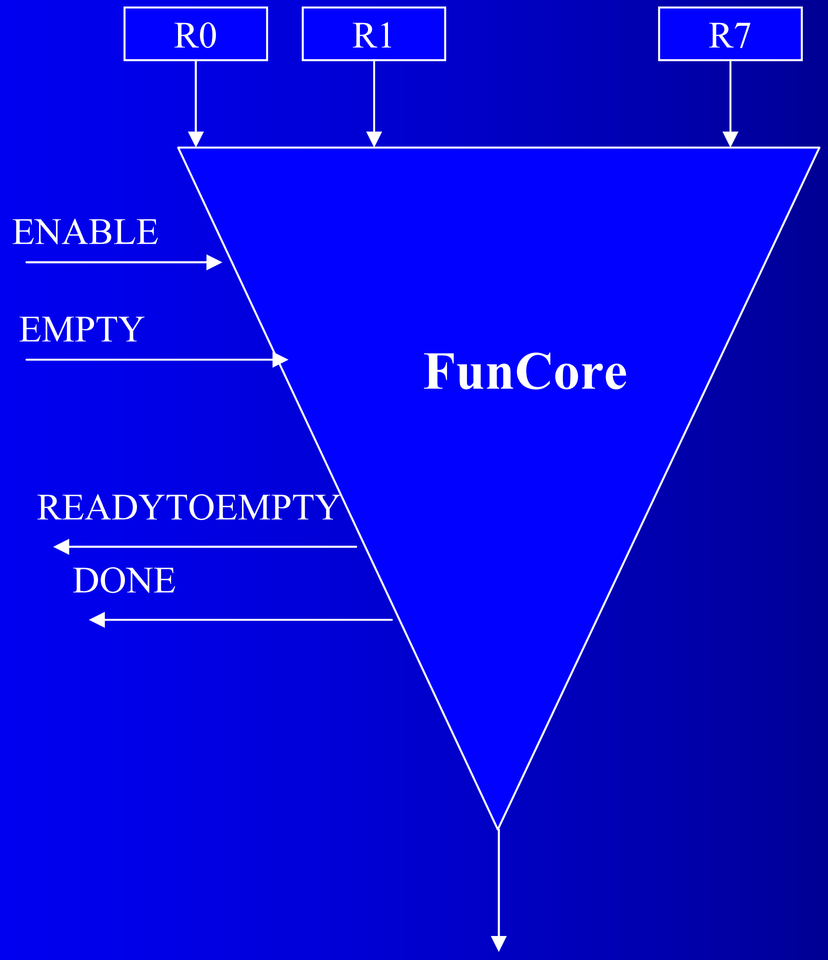


Conditional Functional Cores

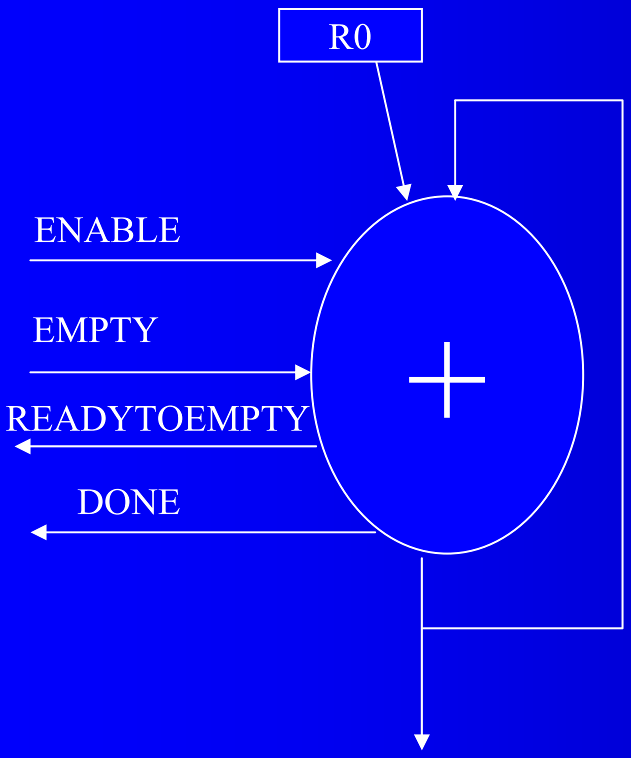


If (R0 > R1) THEN Y = R2 * R3

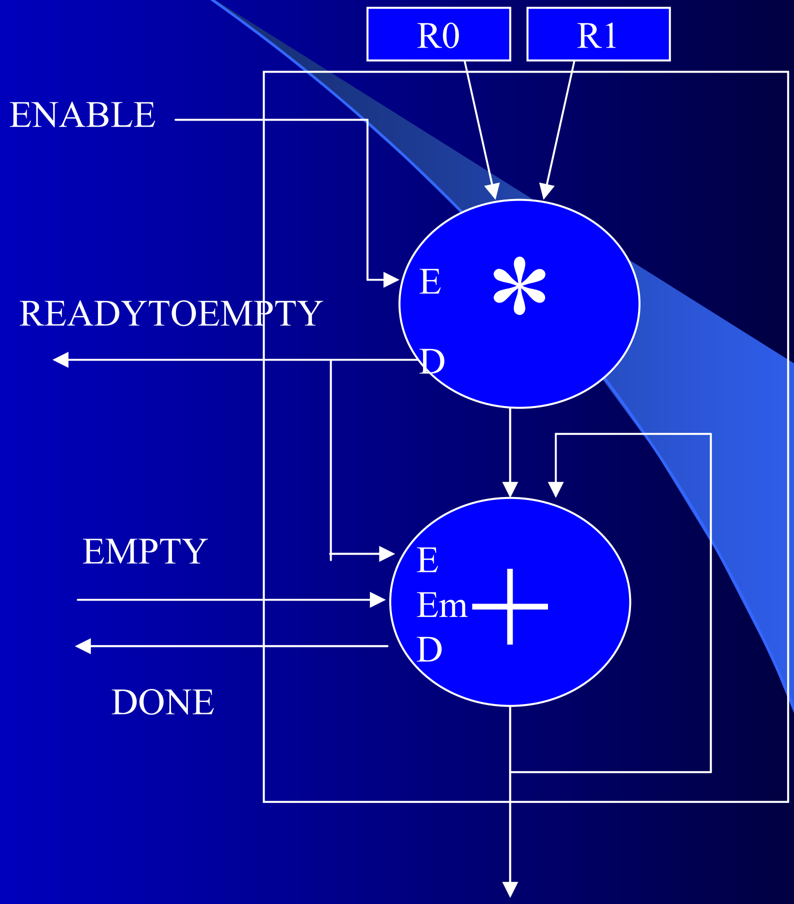
Functional Cores with an Accumulator (Type II)



Sample Functional Cores with with an Accumulator (Type II)

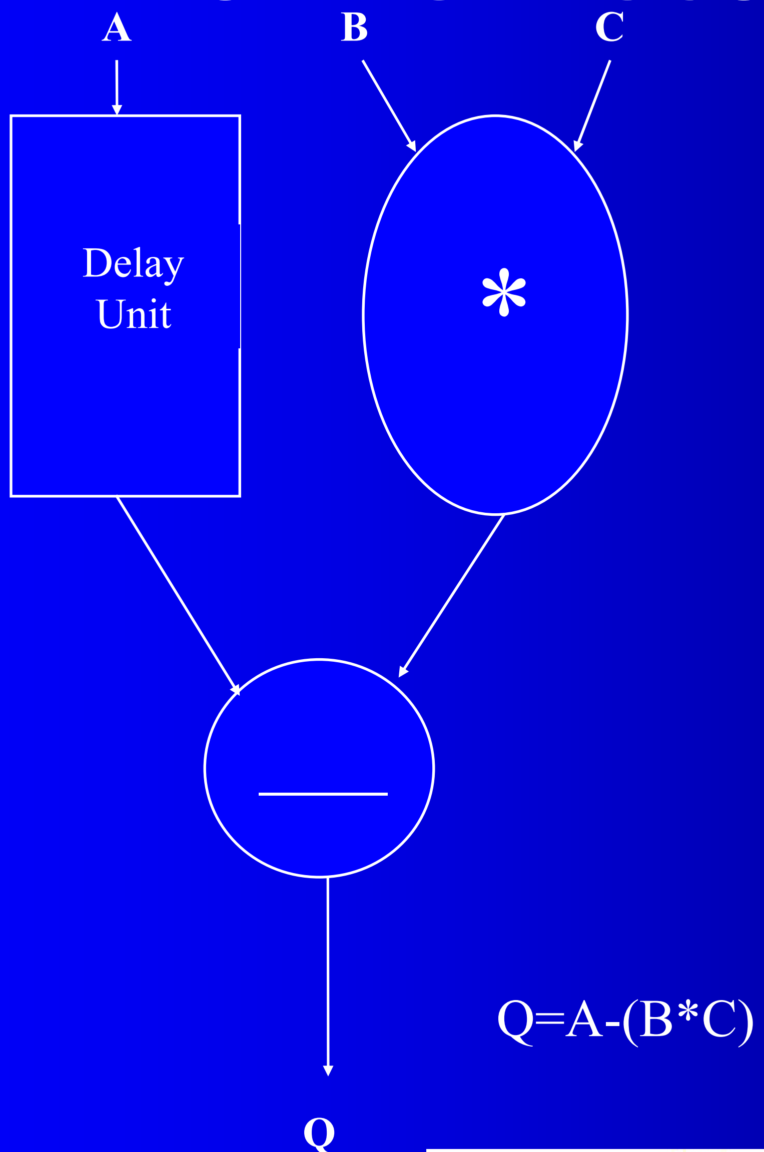


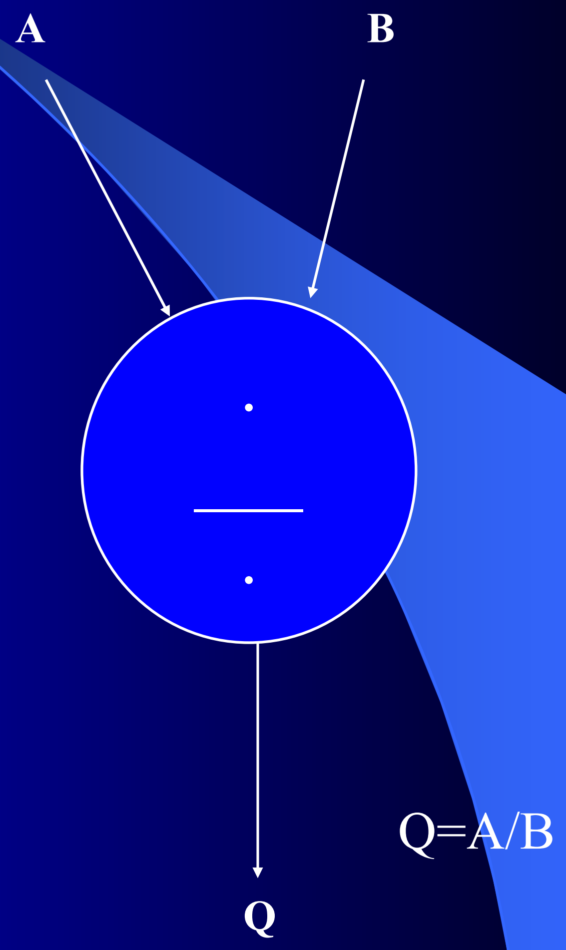
$Q = Q + R0$



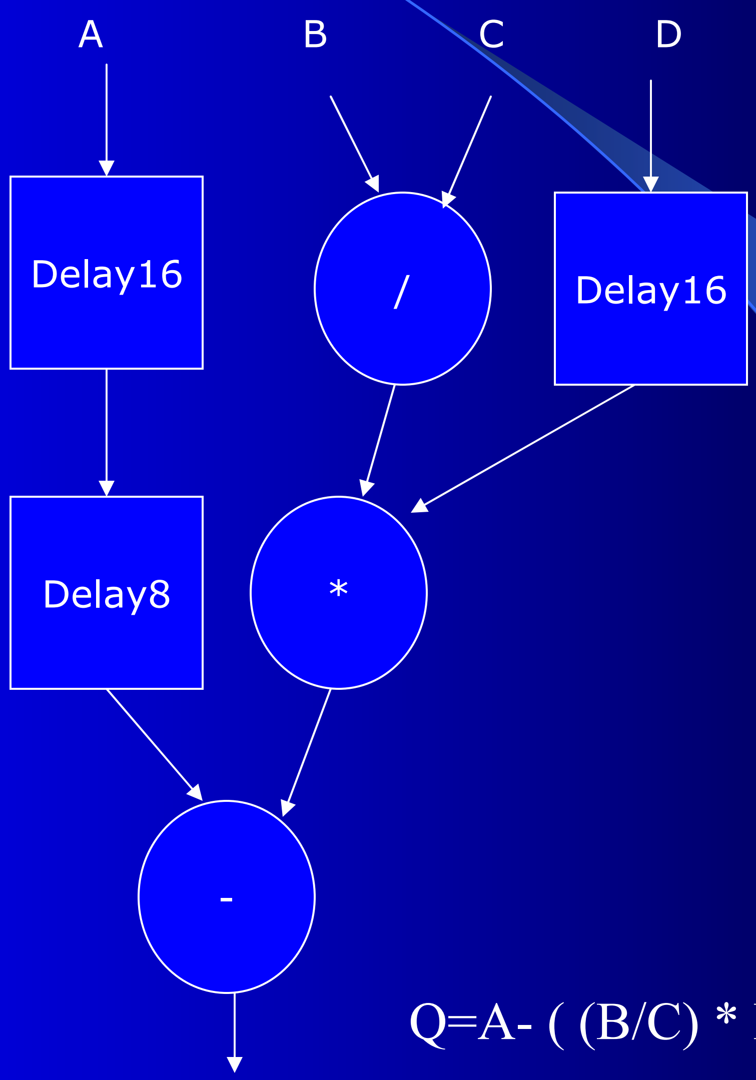
$Q = Q + (R0 * R1)$

Two Functional Cores Used for LU Decomposition





An Alternative Functional Core for LU Decomposition



$$Q=A- ((B/C) * D)$$

Functional Core Performance/CLB Utilization

Core Name	LUTs	Maximum Clock Frequency
FPAdder	526 (1%)	63.9 MHz
FpAccumulator	806 (2%)	46.1 MHz
FpMultiply	1245 (3%)	64.7 MHz
FpDivider	2072 (5%)	56.3 MHz
FPMultiply-Accumulate	2158 (5%)	46.4 MHz

*Xilinx XCV2000E Part, Speed Grade 6, Package FG860

**Available at http://www.imappl.org/~cgloster/rare/vhdl

Functional Core Library (contd)

Core Name	LUTs	Maximum Clock Frequency
Complex Multiply	1025 (2%)	67.0 MHz
Complex Accumulate	1610 (4%)	43.9 MHz
LUCore	3868 (10%)	56.3 MHz
DFTCore	6094 (15%)	39.1 MHz

*Xilinx XCV2000E Part, Speed Grade 6, Package FG860

**Available at http://www.imappl.org/~cgloster/rare/vhdl

Configurable Microprocessor Module Library

Module Name	# of Cores	# of Instructions	LUTs	Maximum Clock Rate
Add-MAC-Module	2	5	4591 (11%)	55.9 MHz
Floc-Module	1	4	4558 (11%)	45.9 MHz
DFT-Module	2	5	37293 (90%)	13.1 MHz

*Xilinx XCV2000E Part, Speed Grade 6, Package FG860

Conclusions / Future Research

- We have demonstrated the effective design and implementation of functional cores and configurable microprocessor modules
  - Modules use standard control/data units
  - Modules can contain several function cores
  - Modules can contain several simple and complex instructions.
- We have developed configurable microprocessor modules that can be automatically selected by a compiler when needed.
- Future Work:
  - Enhance hardware modules to perform multiple simultaneous operand fetches.
  - Modify configurable microprocessor module design to contain an instruction pipeline.
  - Implement control unit as micro programmed control instead of hard-wired control.
  - Consider the addition of branch instructions to the module.