Reconfigurable Computing

Fundamentals, Architectures, and Tools

Dr. Andreas Koch
TU Braunschweig
Germany
Dept. for Integrated Circuit Design (E.I.S.)
koch@eis.cs.tu-bs.de

Presentation Structure

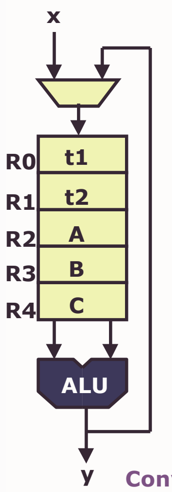
- Fundamentals
- Motivation
- Architectures
- Design Flows
- COFFEE BREAK
- Sample Applications
- Practical Tips & Tricks
- Current Technology
- Summary

Comparison of Paradigms

- Conventional compute unit
  - Temporal distribution of computation
  - Reuse of area for different operations
  - Only one operation per time step
  - Controlled by variable software
  - Universal
- Reconfigurable compute unit (RCU)
  - Spatial distribution of computation
  - Dedicated area for each operator
  - Multiple operations per time step
  - Controlled by fixed controller
  - Made universal only by ability to *reconfigure*

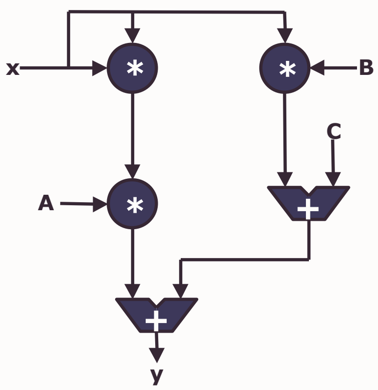
Temporal versus Spatial Computation

$\boxed{y = A\,x\text{^}2 + B\,x + C}$




```
t1 := x
t2 := A * t1
t2 := t2 + B
t2 := t2 * t1
y  := t2 + C
```

Conventional Processor



Reconfigurable Processor

Based on figure by André deHon

Continuum of Architectures

- Many architectural choices between
  - Pure temporal distribution
  - Pure spatial distribution
- Examples
  - Superscalar processors
    - Multiple compute units per time step
    - Increased degree of parallelism
  - Area-constrained reconfigurable processors
    - Reuse of area by reconfiguration
    - Reuse of area by shared operators
    - Non-pipelined multi-cycle operations
    - Decreased degree of parallelism

Terminology I

- *Configurability*
  - Ability to structurally adapt compute unit to specific problem(s)
  - Increased spatial distribution of computation
    - Hardware accelerators for software operations
  - Includes configurable processors (extensible ISA)
    - Tensilica Xtensa and ARC ARCtangent cores
- *Reconfigurability*
  - Ability to configure *after* hardware has been deployed
- *Dynamic reconfiguration*
  - Reconfiguration during algorithm execution
  - Also called run-time reconfiguration (RTR)

Terminology II

- ***Programming***
  - **Vary behavior while preserving structure**
  - **Example: Writing parameters to HW registers**
- **Discussion**
  - **FPGAs generally support only reconfigurability**
    - **No dynamic reconfiguration (far too slow)**
  - **Hybrid approaches in practice**
    - **Configurable processors may have an RCU**
      - Experimental ST device with Xtensa + FLEXEOS
    - **Hardwired ASICs may allow reconfiguration of individual logic elements**
      - eASIC's eASICore with vCells
    - **Program new data values into RCU registers**
      - Often much faster than reconfiguration

Terminology III

- ***Granularity***
  - **Extent of the functionality of individually configurable elements**

fine

- **Transistor pairs (rare, was Crosspoint)**
- **Lookup-Tables (very common)**
- **PLD-like (e.g., Altera, Lattice)**

"FPGAs"

- **ALUs**
  - 4b (Elixent)
  - 8b (MIT MATRIX)
  - 24b (PACT)
  - 32b (Chameleon)

"network processors"
"adaptive processors"

- **Complete processors**
  - 16b (picoChip)
  - 32b (MIT RAW)

coarse

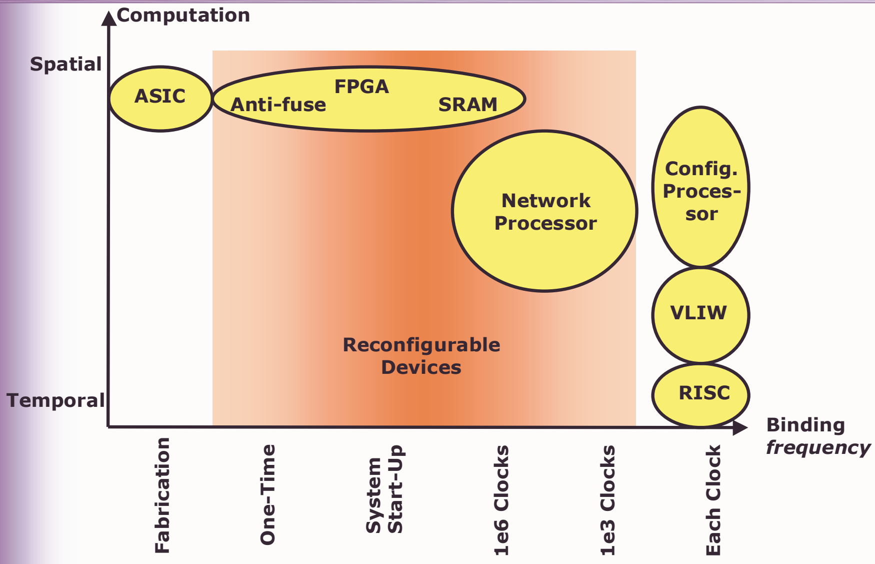
Terminology IV

- *Binding interval*
  - Shortest interval between changes in function
    - May be theoretical (e.g., infinity for ASIC)

Granularity and Binding

- Binding interval often depends on granularity
  - Coarser granularity
  - ➨ Shorter binding interval (less configuration data)
- Shorter binding intervals
  - Better reuse of reconfigurable resources
    - Allow spatial implementation of more kernels
  - Continuous single cycle reconfiguration
    - Tricky: Millions of CMOS transistors switching simultaneously ➔ Poof!
- But match both to application (domain)
  - Single large kernel ➔ reconfigure just at start-up
  - Bit-oriented cryptography ➔ use fine granularity

Terminology IV cont'd.



Based on figure by André deHon

Motivation

- Today's CPUs and DSPs seem pretty ...
  - fast
  - cheap
  - low-power
  - easy to program

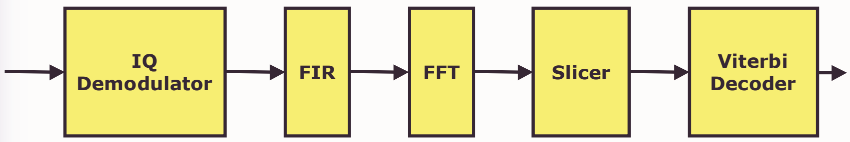
★ So why consider anything else?

RCU Performance

- Early success: gene sequence matching
  - 1993: SPLASH-2 beats MasPar MP-1 by 1300x
- Many successes in cryptography
  - 1999: IDEA encryption 12x CPU, 1.4x ASIC
  - 2001: World record RSA decryption (600Kb/s)
  - 2001: DES encryption 2x ASIC (13.3 Gb/s)
- Digital signal processing
  - "10x-1000x practically achievable over DSPs"
    -- Ray Andraka, FPGA DSP Guru
  - FPGA vs DSP
    - Altera Stratix @ 250 MHz: 56.0 GMACs
    - TI 32064Cx @ 600 MHz: 4.8 GMACs
    - ⋆ But raw performance numbers may be misleading!

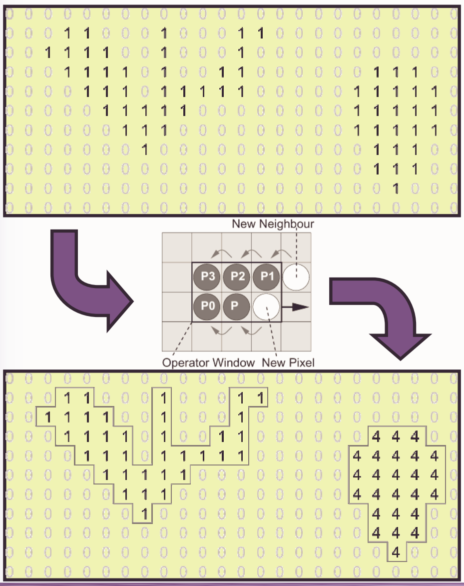
Performance cont'd.

- Application-level cost-performance
  - Full analysis available from BDTI
- OFDM receiver



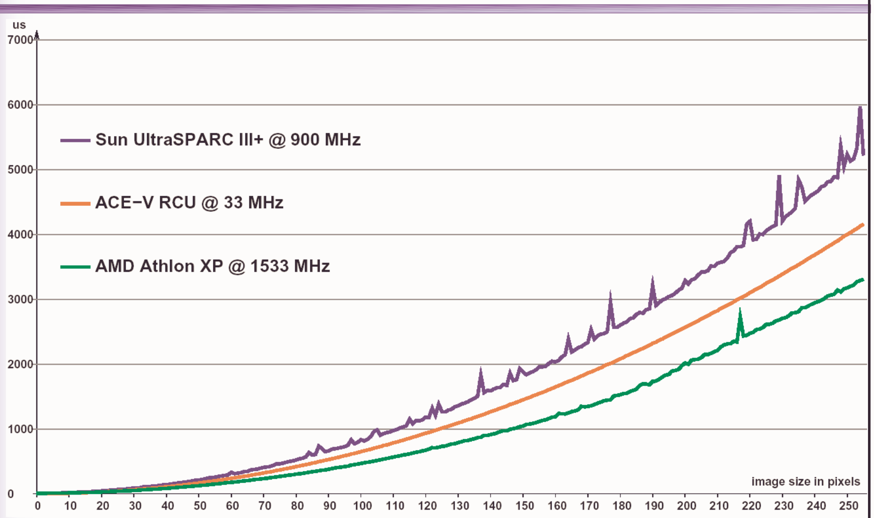
- Motorola MSC8101 DSP @ 300 MHz
  - << 1 channels, \$140 ➧ ~\$500 per channel
- Altera Stratix 1S20-6 FPGA
  - >12 channels, \$325 ➧ ~\$10 per channel

Performance cont'd



- Sample application
  - Label objects in b/w images
  - Scan image using operator window
- Student design
  - CS undergraduate
- Tool flow
  - Verilog HDL
  - C

Performance cont'd.



- Application fits in XC2S100E: US$ 23 part

RCU Power Consumption

- "Power-dissipation of a well-executed FPGA design is typically about 20% of the power consumption of a software-based system operating at the same sample rate"
  -- Ray Andraka, EDN Oct 3, 2002
- Experimental low-power FPGAs do better
  - BWRC LP_PGAII: up to

    70x reduction in energy

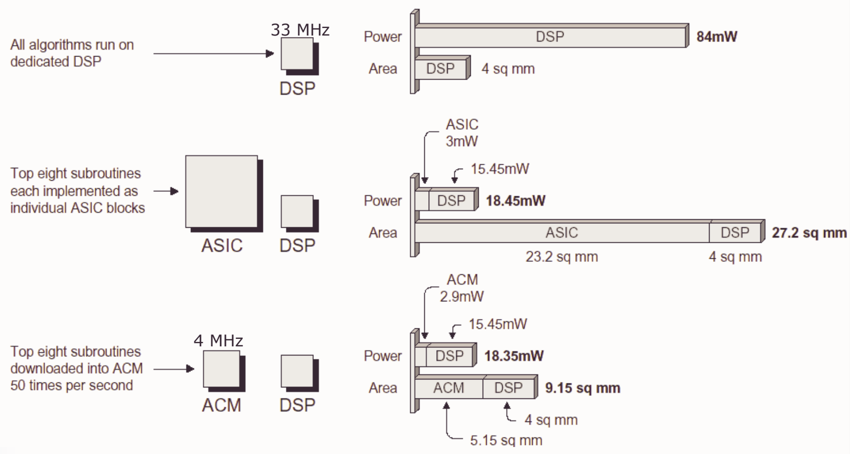
    over equivalent Xilinx XC4005XL part
  - In reconfigurable SoC Maia for VSELP encoding:

    ~20x reduction in energy

    over 2.5V ARM8@120MHz

Power cont'd.

- QCELP encoder on QuickSilver ACM



Courtesy QuickSilver Tech.

RCU Flexibility

- Reconfigurability allows early implementation start
  - Despite fluid standards
- "Interoperability insurance"
- Improve performance after deployment
  - Experience gained from field use
- Allow use of completely new algorithms
  - Limited only by RCU capabilities (area, speed)

➨ Reconfigure to new application versions

- Even better than configurable CPUs/DSPs
  - Have to get custom instructions right the first time

Flexibility cont'd.

- Example
- TSI TelSys equipment for satellite comm.
  - High-rate communications
  - Signal processing
  - Multiple
    - Network protocols
    - Data formats

➨ Use standard hardware platform
  - ACEcard
    - Sun uSPARCIIep RISC + 2x Xilinx XC6264 FPGAs
  - ACE2card
    - Sun uSPARCIIep RISC + 2x Xilinx XC4085XL FPGAs

Architectural Efficiency

- ❑ Moore's Law still holds:

  2x transistors / 18 months

- ❑ Unfortunately, this does *not* guarantee:

  2x performance / 18 months

- ❑ Example: Intel Pentium III CPU
  - ❍ 1999: 500 MHz, 9.5M transistors, ext. L2 cache
    - 20.6 SPECint95, 14.7 SPECfp95
  - ❍ 2000: 1000 MHz, 28M transistors, int. L2 cache
    - 46.8 SPECint95, 32.2 SPECfp95
  - ❍ Sounds good: 2.3x int, 2.2x fp, but ...
    - 2x clock freq and 3x transistors to get there

Efficiency

- ❑ Current fab processes: 300 M transistors
- ❑ What to do with this much real estate?
  - ❍ Larger caches
    - HP PA-RISC 8700: 1.5MB L1 cache on-chip
    - SPEC benchmarks execute completely in cache
  - ❍ Higher integration
    - On-chip memory controllers
  - ❍ Multiple processors on-chip
    - HP PA-RISC 8800: 2x PA-RISC 8700
- ✗ ... but not much architectural innovation
- ❑ Idea: Spend some transistors on RCU

Efficiency cont'd.

- ❑ **Transistor budgeting**
  - ❍ **Example: Xilinx Virtex 1000 FPGA**
    - **75 M transistors / 1 M gates RCU capacity**
  - ❍ **Much denser architectures exist**
- ❑ **But even smaller RCUs can still be useful**
  - ❍ **B/W image labeling: ca. 100 K gates**
  - ❍ **From EEMBC benchmarks:**
    - **Add custom instructions to Tensilica Xtensa**
      - Use 22K gates: 37x performance "telecom"
      - Use 200K gates: 23x performance "consumer"
    - **Add custom instructions to ARCcore ARCtangent**
      - Use 58K gates: 40x performance "telecom"
      - Use 113K gates: 18x performance "consumer"

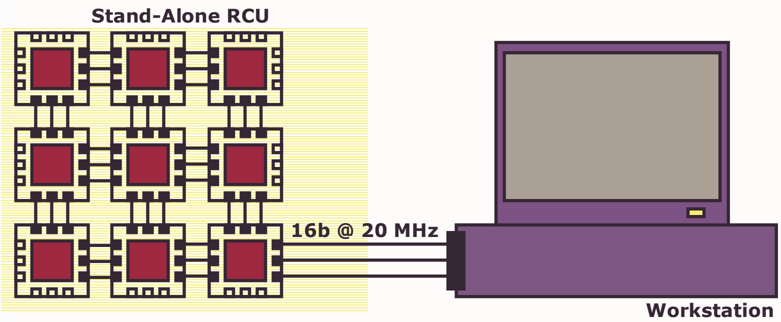
Economics of Fabrication

- ❑ **More transistors per chip, but ...**
  - ❍ **Tool flow challenged (timing closure etc.)**
  - ❍ **Fab on advanced process *extremely* expensive**
    - **Higher cost of masks, more masks per chip, ...**
- ❑ **Advanced fab technology only for**
  - ❍ **Cost-insensitive applications**
    - **Requirements dominate, "it just has to work"**
  - ❍ **High-volume applications: CPUs and DSPs**
    - **Multiple uses for each device are a *must***
    - **But reduced performance, power, efficiency, ...**
- ❑ **RCUs might fill the gap**
  - ❍ **FPGAs already act as process drivers**

System Architecture

- ❑ How to integrate an RCU into a system?
- ❑ RCU does *not* automatically imply FPGA!
  - ❍ FPGAs have been around longest
  - ✗ … but are far from perfectly suited as RCU:
    - Fine granularity ./. word-oriented applications
    - Glacial configuration speed
      - Order of 100ms for large devices
      - Precludes dynamic reconfiguration
  - ❍ Recent improvements
    - Heterogeneous blocks (RAMs, multipliers)
    - On-chip processors
  - ➨ … aid in improving system integration, but not the idea of dynamic reconfiguration
      - Inefficient use of silicon area

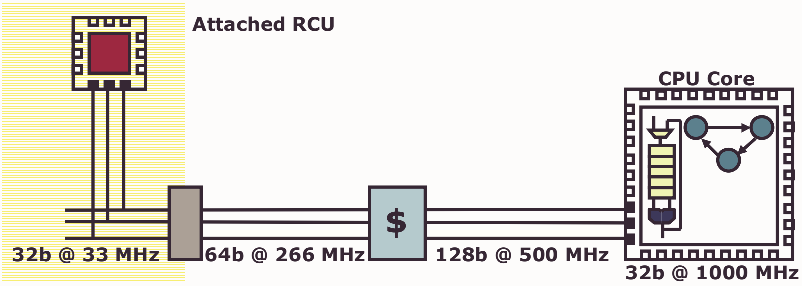
Stand-Alone RCU



- ❑ Example: ASIC Emulation
  - ❍ Attached via SCSI
  - ❍ 112 M gates reconfigurable capacity
    - RCU weighs 1.1t
    - 12KW 350V three-phase power
- ❑ Very limited set of suitable applications

Based on figure by Scott Hauck

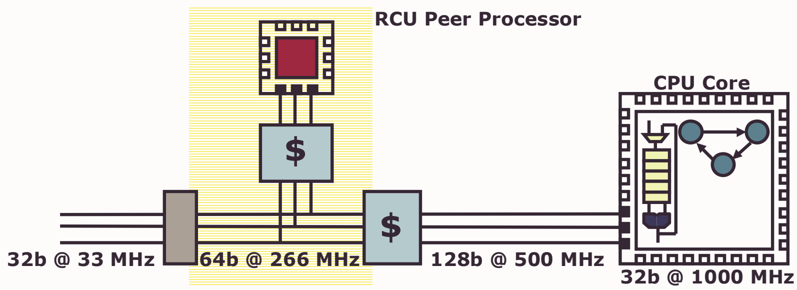
Attached RCU



- ❑ **Attached to peripheral busses**
  - ❍ PCI, VME, SBus, ...
  - ❍ Standard busses, RCU easy to deploy
  - ❍ Most common method of RCU integration
- ❑ **Better than stand-alone, but still slow**
  - ❍ PCI write latency: 10 clocks, read: 30 clocks

Based on figure by Scott Hauck

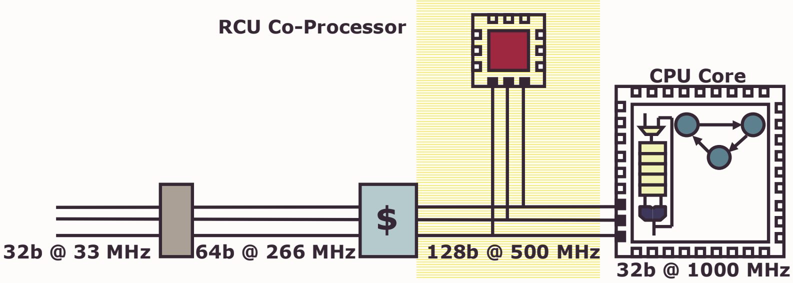
RCU Peer Processor ("SMP")



- ❑ **Equal partner to CPU (SMP-like)**
  - ❍ Much higher bandwidth, lower latency
  - ❍ RCU implementation of multi-processor bus protocols (133 MHz should be achievable)
    - Interrupt handling, cache coherency, ...
  - ❍ RCU could be retrofitted into standard SMP boards
  - ❍ No practical realization yet (?)

Based on figure by Scott Hauck

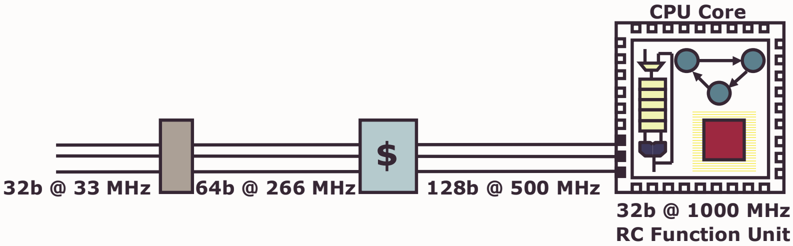
RCU Co-Processor



- ❑ Attached to internal processor bus
  - ❍ Shares cache with processor (possibly only L2)
  - ❍ No (or fewer) coherency issues
  - ❍ More bandwidth, less latency
- ❑ Implementation based on standard cores
  - ❍ UCB GARP (=custom RCU + MIPS core)

Based on figure by Scott Hauck

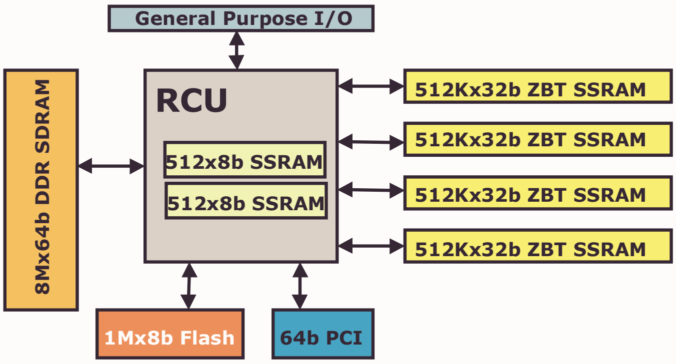
RC Function Units



- ❑ RCU as function unit
  - ❍ Directly integrated into the processor datapath
  - ❍ Very low latency
  - ❍ Generally: Limited bandwidth (data starvation)
    - Operates only on 2-3 registers per instruction
      - Some exceptions: OneChip-'98 has memory port
    - Can still be useful: PRISC-1 gains 22% on SPECint92
- ❑ Needs custom or configurable processor core

Based on figure by Scott Hauck

I/O and Memory



- **Heterogeneous memories**
  - On-chip / off-chip
  - Multi-bank / multi-type
- **Configurable I/O system**
  - Sometimes expandable by daughter board

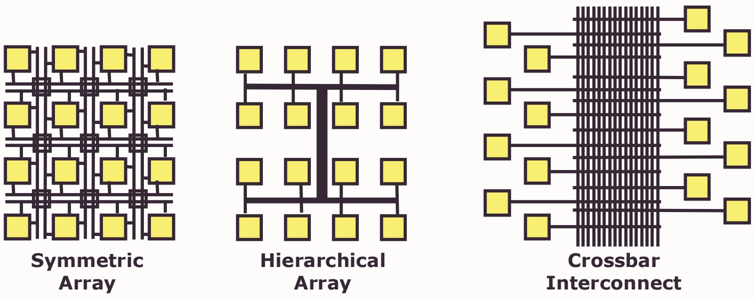
Effect on Applications

- **Suitable applications depend strongly on degree of coupling**
  - Table shows typical RCU execution times

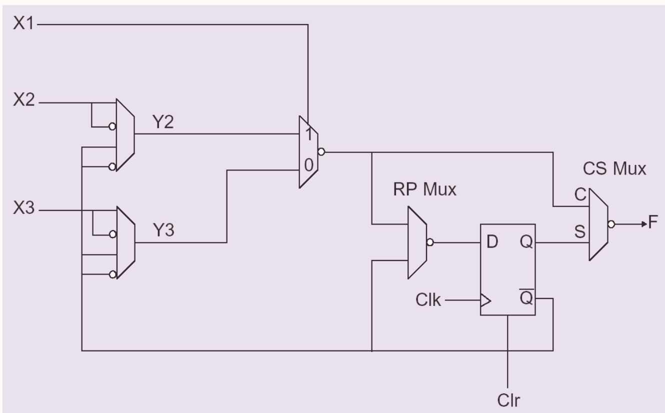
Application / RCU Type	Minimal effective computation time	Data I/O rate
Stand-Alone	Very long *(~10s)*	Very low
Attached	Long *(~10ms)*	Medium
Peer Processor	Medium *(~100us ?)*	High
Co-Processor	Short *(~1us)*	High
Function Unit	Very short *(~10ns)*	Low

Device Architecture

- General idea
  - Configurable interconnection network
  - Configurable function blocks
- Many variations possible!
- Example: Interconnection networks



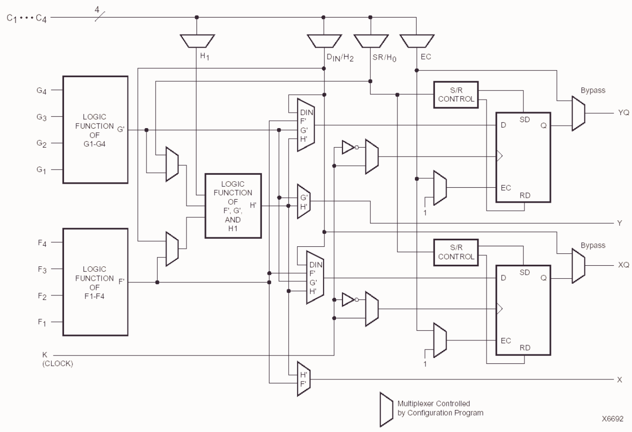
Fine-Grained Block



- Xilinx XC6200 logic block
- Realizes
  - Any 2-input function
  - Some 3-input functions

Figure from Xilinx Datasheet

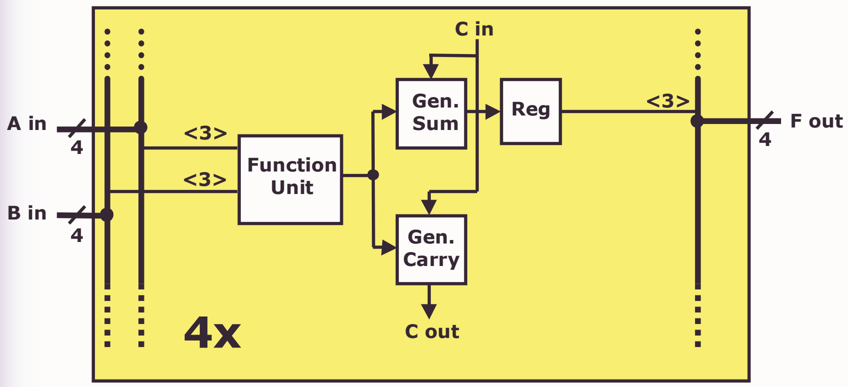
Medium-Grained Block



- ❑ **Xilinx XC4000 block**
  - ❍ **Two arbitrary 4-input functions**
  - ❍ **Some wider functions (e.g., 2b add/sub)**

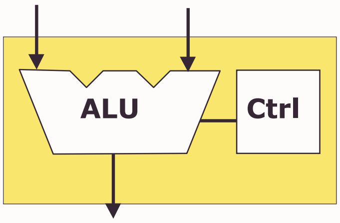
Figure from Xilinx Datasheet

Coarse-Grained Block



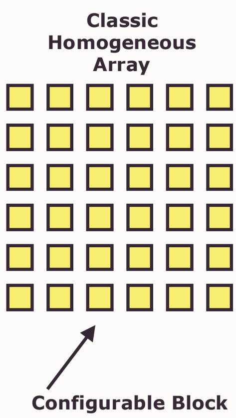
- ❑ **HP Labs CHESS (now Elixent D-Fabrix)**
  - ❍ **4b ALU**
  - ❍ **Logic and simple arithmetic (add, sub)**
  - ❍ **Function controllable by another block at run-time**
  - ❍ **Example: JPEG encoder takes 512 ALUs of area**

Very Coarse-Grained Block



- **PACT XPP ALU block**
  - 24b and (12b, 12b) split-operation
  - Logic, arithmetic including multiplication
  - Automatic synchronization for
    - Data flow
    - Partial run-time reconfiguration

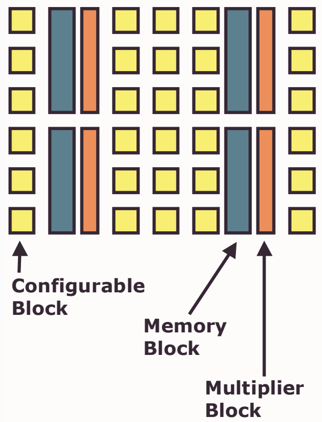
Homogeneous Arrays



- **Traditional FPGAs are *homogeneous***
  - Single type of configurable element
    - Possibly multi-functional
      - Logic or RAM mode
    - Composed to assemble any digital function
  - ✔ Advantages
    - Simpler tools
    - Simpler device layout
  - ✗ But may be very inefficient, for
    - Multipliers
    - Larger memories

Heterogeneous Arrays

Modern Heterogeneous Array



- ❑ *Heterogeneous* devices
- ❑ Embedded hardwired blocks
  - ❍ Fast multipliers
  - ❍ Larger memories
  - ❍ Even complete processor(s)
  - ❍ Clock Management
  - ❍ Specialized I/O interfaces
- ✔ Higher performance
- ✔ More efficient area usage
  - ✗ Only when blocks are used!
- ✗ Increased tool complexity
  - ❍ Must obey additional constraints

Architecture Trends

★ With this prior development, what's next?

❑ Three broad approaches have become visible

❶ System FPGAs

❷ Reconfigurable Systems-On-Chip (rSoC)

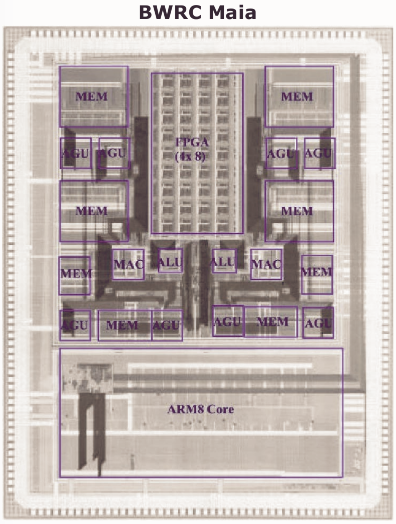
❸ Specialized devices for adaptive computation

System FPGAs

- ❑ Higher capacity reduces number of devices on board
  - ❍ Xilinx XC2V8000: 8 M configurable logic gates
- ❑ On-chip features for improved system-level density
  - ❍ Integrated processor(s)
    - Up to 4x PowerPC 405 cores in Xilinx Virtex II Pro devices
  - ❍ Digitally controlled impedance
    - Replaces board-level termination resistors
- ✘ But reconfiguration is still rather slow
  - ❍ At best ~50ms for large devices
- ➨ Not really aimed at *reconfigurable* computing
  - ❍ Infrequent mode switches
  - ❍ Soft-hardware updates

Reconfigurable SoCs (rSoC)

BWRC Maia

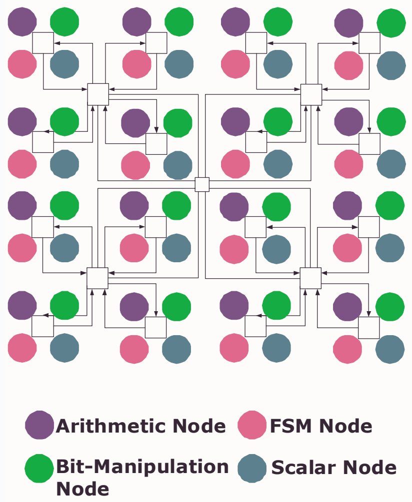


- ❑ Heterogeneous SoCs including RCU(s)
  - ❍ Customized for application domain(s)
    - But still flexible to handle new developments
- ❑ True reconfigurable computing possible
  - ❍ High on-chip bandwidth allows fast configuration
    - 500us for 200 K gates on fine-grained fabric
      - M2000's FLEXEOS IP
    - 33us for 128 ALUs on a very coarse-grained fabric
      - PACT's XPP128-ES, core now available as IP

Figure from Varghese George

Adaptive Computing Devices

QuickSilver ACM



- ❑ Specifically built to efficiently *compute*
- ❑ Single clock cycle reconfigurability
- ❑ Ultra low power
- ❑ Heterogeneous array
- ❑ Example: QuickSilver ACM
  - ❍ >57.000 reconfigs/s for CDMA2000 Rake finger
  - ❍ 200 MHz ACM vs ASIC
    - CDMA2000 searcher
      - 108x
    - CDMA2000 pilot search
      - 108x
    - W-CDMA searcher
      - 74x

Courtesy of QuickSilver Tech.

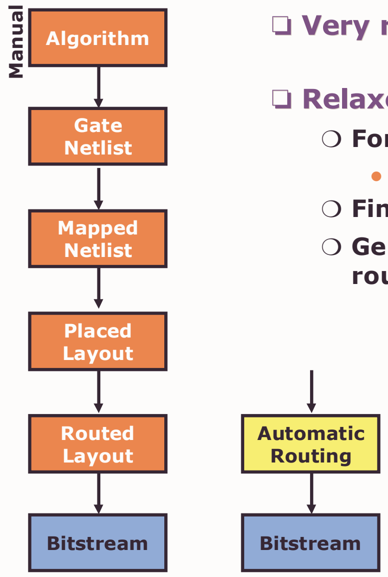
Design Flows for RCUs

- ❑ How to program these contraptions?
  - ❍ Quickly
  - ❍ Efficiently
  - ❍ Correctly
- ❑ Three variables
  - ❍ Cover only hardware or hard- and software
  - ❍ Degree of tool support
    - Fully manual ↔ fully automatic
  - ❍ Input format of algorithm description
    - Related to computation model used
      - Data flow-oriented (many variations)
      - State machines (e.g. Harel diagrams)
      - Imperative (common software languages)
      - Structural (schematics or some HDL style)

Hardware and/or Software?

- Depends on application area
- Scenarions for pure hardware
  - High-speed interfaces
    - Possibly with pre-processing: Collider event detection
  - Glue logic
  - Simple state-machines
    - Traffic lights, vending machines, ... :-)
- Scenarios for combined hardware/software
  - Compute kernels in hardware
    - Small blocks of compute-intensive code
      - Loop nests
    - Often streaming code
      - Array/matrix operations
  - Complex irregular control in software
    - Application and system-wide control
      - Operating system

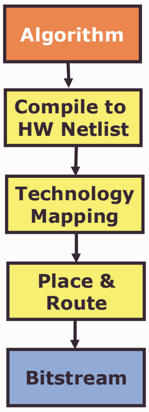
Fully Manual



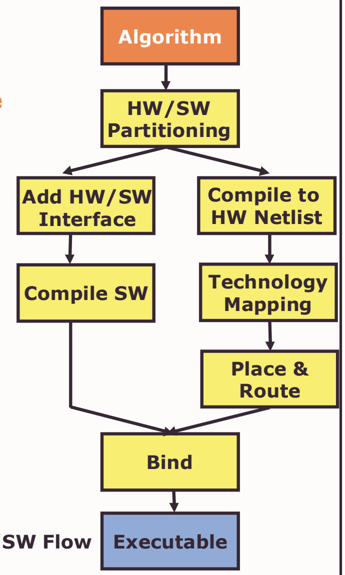
- Very rare in this extreme form!
- Relaxed form sometimes used
  - For high-performance designs
    - "John Henry" approach
  - Finely tuned hard IP blocks
  - Generally relies on automatic routing

Fully Automatic Flows

- ❑ Translate algorithm
  - ❍ … pure HW (limited!)
    - HDL synthesis, Forge
  - ❍ … into HW and SW
    - GarpCC, Nimble-C
  - ❍ Often:
    - Manual partitioning

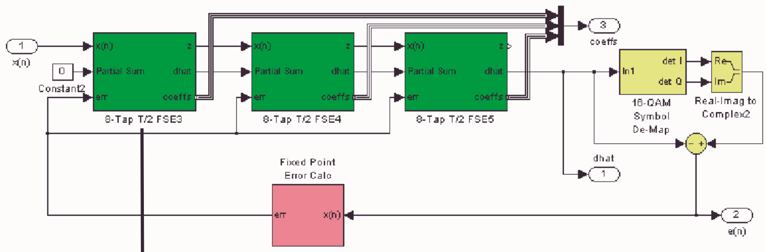


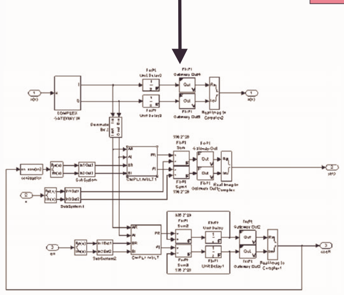
Pure HW Flow



Hybrid HW/SW Flow

Graphical Entry





- ❑ Compose blocks
  - ❍ Hierarchical, parameterized
  - ❍ Ptolemy, SPW, Simulink, …
    - Also circuit schematics
- ❑ Fractionally Spaced Equalizer
  - ❍ Simulink representation
- ❑ Suitable for DSP+telecomm
  - ❍ Or low-level design entry

Figure by C. Dick, H.M. Pedersen

Textual Description

FIR in ANSI C

```
fir(  int input[],
      int coef[], int nCoef,
      int output[], int nOut )
   {
      int i, j;
      int  sum;

      for (j =  0; j < nOut; j++) {
         sum =  0;
         for (i =  0; i < nCoef; i++){
            sum +=  input[j+i] * coef[i];
         }
         output[j] =  sum >> 15;
      }
   }
```

FIR in SilverC

```
void run (void)
{
   fract16 sum;
   loop (int l=0; l<nOut; l++) dataflow {
      sample = input.read();
      sum = 0.0;
      unroll (int i=0; i<nCoef; i++) {
         sum = sum + coefReg[i] * sample[nCoef-i];
      }
      output.write(sum);
   }
}
```

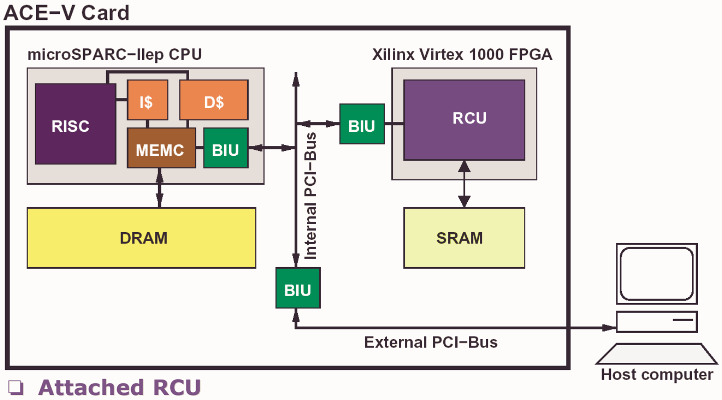
- Very high-level languages: MATLAB
- Conventional high-level languages: C, Java
- Specialized RC languages: TDFC, Handel-C, SilverC,
- Hardware description languages: Verilog, VHDL

Code courtesy of QuickSilver Tech

HDL-based Programming

- Currently the most common way to program RCUs
- Use HDL to formulate the hardware parts
  - Generally at register-transfer level (RTL)
  - Some structural parts to access special RCU hardware
    - Multipliers, multi-port memories, DLLs, ...
- Software parts in high-level programming language (HLL)
  - C, C++, some Java
- Reasonably robust tool support for
  - HLL compilation
  - HDL synthesis
  - Technology mapping, placement & routing
  - Simulation

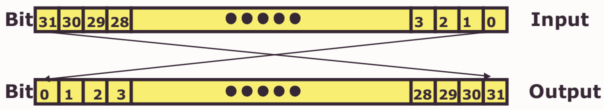
Target Environment: ACE-V



- Attached RCU
  - CPU: 100MHz microSPARC-IIep RISC
  - RCU: Xilinx Virtex XCV1000-4 FPGA
- 64MB DRAM (shared), 4MB SRAM (RC-local)
- On-board Bus: 33MHz 32b PCI
- Custom port of RTEMS 4.0.0 as operating system

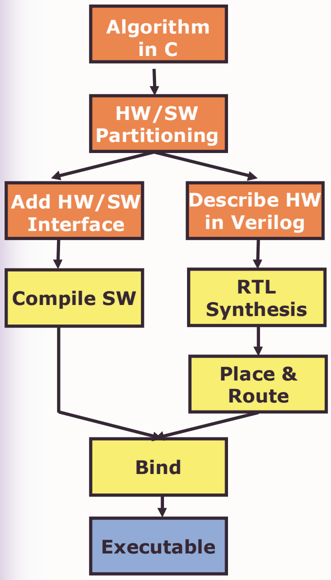
Sample Application

- Practical example
  - Software development
  - Hardware development
  - Hardware/software interfaces
- Application: Reversal of bit order in 32b word



- Three stages
  1. Pure software solution
  2. Slave-mode RCU
     - CPU controls data transfer
  3. Master-mode RCU
     - RCU controls data transfer

Tool Flow



- Manual algorithm description
- Manual HW/SW partitioning
- Manual HW description
- Manual HW/SW interfacing
- Standard SW flow
- Standard RTL-FPGA flow
- Custom binding phase

Pure Software Version

```
...
// Kernel to process all data words
 for (m=0; m < NUM_WORDS; ++m) {
        inword   = inwords[m];
        outword = 0;
        mask     = 1;
        set        = 1 << 31;

        // Bitwise assembly of the processed word
        for (n = 0; n < 32; ++n) {
                if (inword & mask)
                        outword |= set;
                 mask   <<= 1;
                 set     >>= 1;
         }

         // Enter the result in the output array
         outwords[m] = outword;
 }
...
```

- Compute *kernel* of the pure software version
  - See Listing 1 in your handouts
- Performance: 512 Kw in 1449623us = ~1.5s

Slave-Mode Version

```
module user(
  CLK,          // System clock
  RESET,        // System-wide reset
  ADDRESSED,    // High when CPU addresses RCU
  WRITE,        // High when CPU writes to RCU
  DATAIN,       // Data written from CPU to RCU
  DATAOUT,      // Data from RCU to be read by CPU
  ADDRESS       // RCU Address of access (ignored for this application)
);

  // Inputs
  input          CLK;
  input          RESET;
  input          ADDRESSED;
  input          WRITE;
  input  [31:0]  DATAIN;
  input  [23:2]  ADDRESS;

  // Outputs
  output [31:0]  DATAOUT;
```

❑ Slave-mode interface to RCU

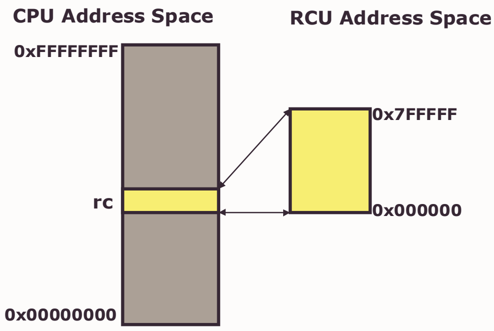
Slave-Mode Compute Kernel

```
reg [31:0] result;          // Register for computation result
reg [31:0] reversed;        // Temporary value

// Always output the result register (independent of address)
assign DATAOUT =  result;

// Compute the bit-reversed version of the current data input value.
// Note: This is a pure combinational block
always @(DATAIN) begin: comb_block
      integer n;
      for (n=0; n < 32; n = n + 1) begin
           reversed[n] = DATAIN[31-n];
      end
end

// Control
always @(posedge CLK or posedge RESET) begin
      // Initialize result register to recognizable magic number (for debugging)
      if (RESET) begin
           result  <= 32'hDEADBEEF;
      // When CPU writes data to RCU, store the reversed word as result
      end else if ( ADDRESSED & WRITE) begin
           result <= reversed;
      end
end
```

Accessing the RCU from SW

```
// Initialize RCU
acev_init();
// Run RCU at a 40 MHz clock
acev_set_clock(40e6);
// Configure RCU with bit-reversal application
acev_load_config(&config_reverse);
// Get pointer to start of RCU address space
rc = acev_get_s0(NULL);
```

- ❑ **RCU API**
  - ❍ Setup
  - ❍ Clock control
  - ❍ Configuration



- ❑ **Memory mapping**
  - ❍ Slave-mode
  - ❍ RCU-CPU space

Slave-Mode Software

```
...
 // Remember start time of actual computation
 start = RTEMSIO_getTicks();

 // Kernel to process data
 for (m=0; m < NUM_WORDS; ++m) {
   // Transfer input data word to RCU
   rc[0] = inwords[m];
   // Fetch reversed result from RCU and store in output array
   outwords[m] = rc[0];
 }

 // The core computation is completed, remember the current time
 stop = RTEMSIO_getTicks();
...
```

- ❑ **CPU controls data transfer**
  - ❍ Write data word to RCU for processing
  - ❍ Read processed word from RCU
  - ❍ Details see Listing 2 in handouts
- ❑ **Performance: 512 Kw in 825365us = ~0.8s**

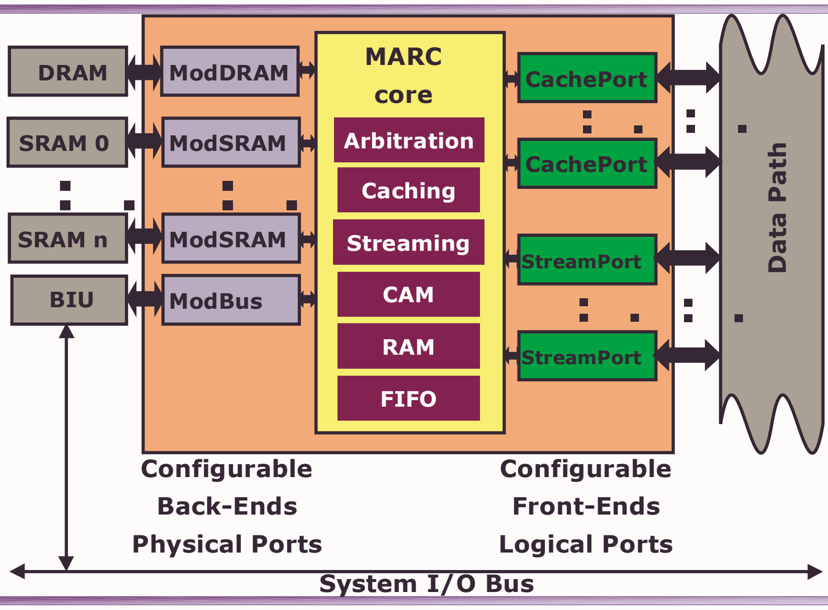
Evaluation

- So far, so good:
  - 40 MHz RCU beats 100 MHz RISC for computation
    - ... ignoring RCU configuration overhead of ~0.9s
- Can we do better?
  - Computation looks pretty tight already
  - But how about communications overhead?
- Measurements
  - Shortest time between read and write accesses
    - 50 RCU clocks
  - Longest time between read and write accesses
    - 694 RCU clocks
  - ➨ Slave-mode is extremely inefficient!
    - Due to PCI sub-system (BIUs in uSPARC and RCU)

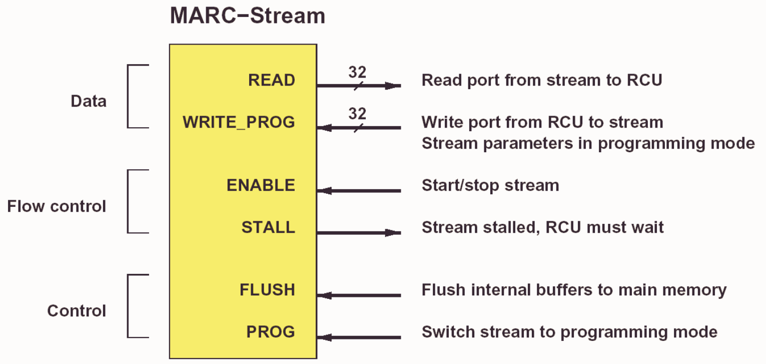
Master-Mode Solution

- Idea
  - Avoid quick read/write direction changes
  - Implement data transfer control in hardware
  - RCU can now independently access main memory
- Protocol engine must be implemented
  - Should exploit burst transfers
    - Requires local buffering
  - Buffer architecture depends on access patterns
    - Irregular: Cache
    - Regular: FIFO
  - Should be reusable
- ✘ ... the problem is becoming complicated
  - ➨ *M*emory *A*rchitecture for *R*econfigurable *C*omputers

MARC IP

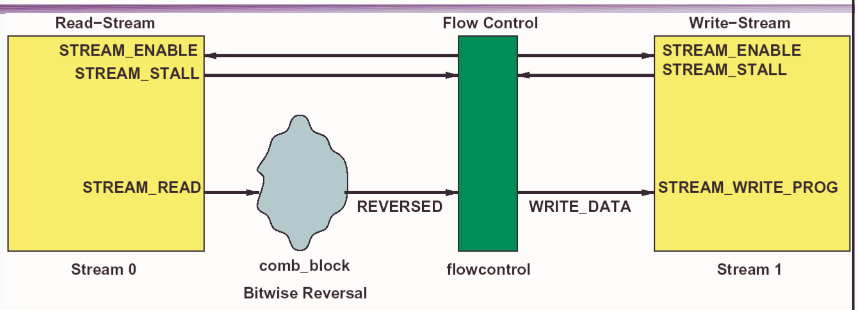


MARC Stream Interface



- Number of streams *configurable*
  - Priority arbitration, but no inter-stream coherency
- Parameters run-time *programmable*
  - Start address, length, stride, width, read/write

Coupling MARC Streams



- **Idea**
  - **Link read and write streams**
  - **Interpose computation**
- **Flow-control mechanism required**
  - **Stop read stream if write stream stalls**
  - **Stop write stream if read stream stalls**
  - **Use forward / backward pressure concept**

Master-Mode HW Interface

```
module user (
        // *** Global signals
        CLK,                    // System clock
        RESET,                  // System-wide reset

        // *** Slave interface
        ADDRESSED,              // High when CPU accesses RCU
        WRITE,                  // High when CPU writes data to RCU
        DATAIN,                 // Data written from CPU to RCU
        DATAOUT,                // RCU output data readable by CPU
        ADDRESS,                // Adress, used both by RCU and CPU
        IRQ,                    // Set high for RCU to interrupt CPU

        // *** Interface to MARC streams
        STREAM_READ,            // Read data bus from MEM to RCU
        STREAM_WRITE_PROG,      // Write data bus to MEM and MARC programming
        STREAM_STALL,           // Per-stream stall signals
        STREAM_ENABLE,          // Start/stop signals for streams
        STREAM_FLUSH,           // Set high to flush write streams to MEM
        STREAM_PROG             // Set high to switch stream to programming mode
       );
```

- **All stream signals aggregated into wide busses**

Master-Mode Hardware

- See Listing 4 (rather long) in your handouts
- comb_block inserted into streams
  - Line 131-139 Bit-wise reversal of passing data words
- RCU starts in slave mode to accept parameters
  - Line 152-162 Start address of input data in main memory
  - Start address of output data in main memory
  - Number of words to process
  - A command to start execution
- RCU-internal controller FSM takes over
  - Line 168-211 MARC streams are appropriately programmed
  - Line 212-234 Streams are started, data is being processed
  - Line 224-229 On end-of-read-stream, flush write stream
    - Force internal FIFOs into main memory
  - Line 242 Indicate completion by interrupt to CPU

Master-Mode Software

```
...
// Handler for RCU-initiated interrupts
void
irq_handler() {
        // Ask RCU to deassert interrupt (any read to RCU-space will do)
        int volatile foo = rc[0];

        // Mark RCU operation as complete.
        // Execution continues in main() after acev_wait(), Line 83
        acev_mark_done();
}
...
void
main() {
    ...
        // Register handler function for RCU-initiated interrupts
        acev_irq_handler(irq_handler, NULL);
        // Mark RCU status as `operation in progress'
        acev_mark_busy();
    ...
        // Program this run's parameters into RCU
        rc[REG_SOURCE_ADDR]   = inwords;     // Start address of input data in memory
        rc[REG_DEST_ADDR]     = outwords;    // Start address for output data in memory
        rc[REG_COUNT]         = NUM_WORDS;   // Number of data words to process
        rc[REG_START]         = 1;           // Send start command to RCU

        // Wait for RCU execution to complete (indicated by interrupt, line 32)
        // CPU could continue operation in parallel
        acev_wait();
    ...
```

Evaluation

Approach	RCU Clock [MHz]	RCU Size [Slices]	Computation Time [us]	Speedup vs. Pure SW
Pure Software			1449623	1.00
Slave-Mode RCU	40	116	825365	1.76
Master-Mode RCU	25	1369	109933	13.19

Slices available on XCV1000: 12228

- Master-mode is considerably more efficient
- Despite of ACE-V misfeatures
  - All memory accesses via PCI
  - Faulty off-chip handshaking
    - Pin not connected on PCB
    - Limited burst length
    - Limited clock speed

Automatic HLL Compilation

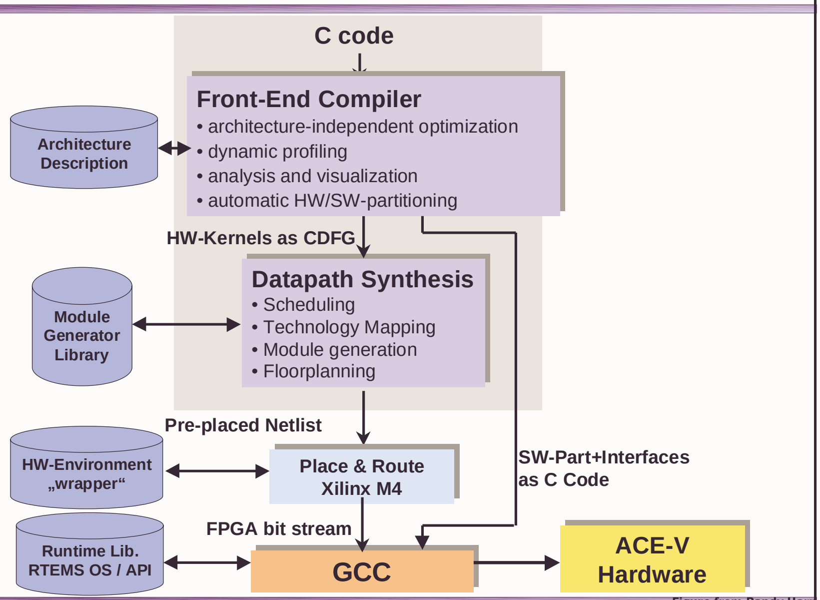


Figure from Randy Harr

Example Program

```c
void
main(int argc, char *argv[])
{
  int i, j, k;

  // Integer value of the first command line parameter
  j = atoi(argv[1]);
  // Integer value of the second command line parameter
  k = atoi(argv[2]);

  for (i = 0; i < k; i++)
    {
     j = j * 13;
     if (j > 1000000)
       printf("j=%d too large in loop i=%d\n", j, i);
    }

  printf("result: j = %d\n", j);
}
```

Sample execution

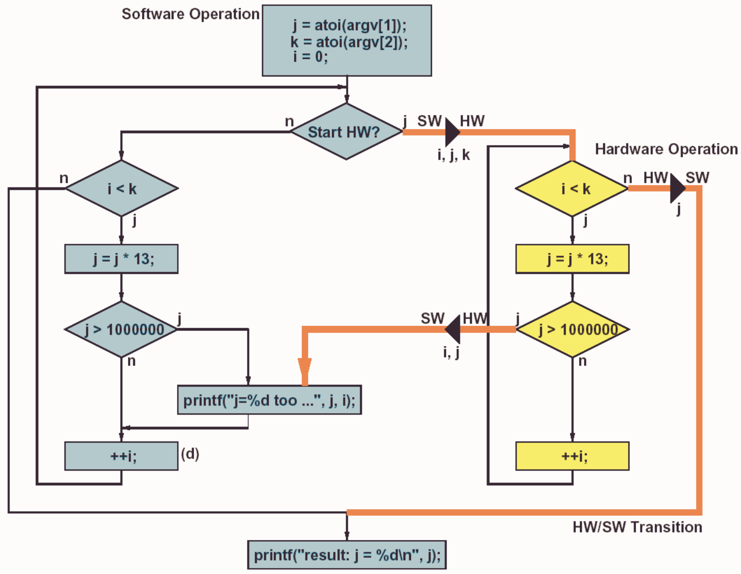
```
$ ./a.out 10 5
j=3712930 too large in loop i=4
result: j = 3712930
```

- Compute j * pow(13, k)
  - Check for an overflow condition, print message

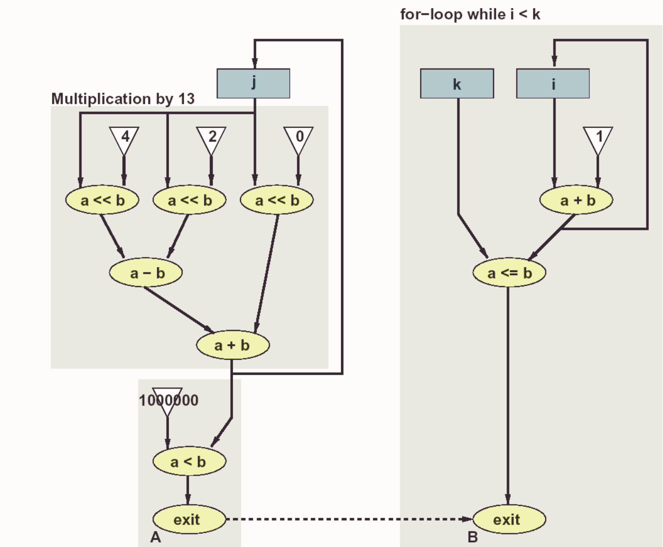
HW/SW Partitioning

- Dynamic profiling identifies kernel
  - Problem: printf() not realizable in hardware
- Most tools give up here
  - Maybe inform the programmer to make a change
- Alternate approach
  - Determine how often the condition occurs in fact
    - Data dependent!
  - If sufficiently infrequent, hardware execution might still be useful
    - But have to handle case if it *does* occur
  - ➨ Manage both HW and SW versions of the kernel

HW/SW Execution

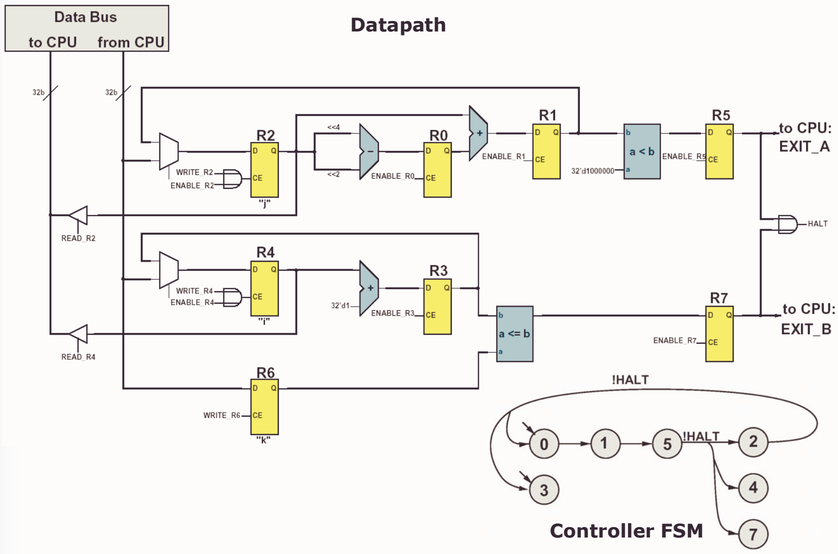


Control-Data Flow Graph



Detect overflow when temporary value > 1000000

Hardware Mapping



HW/SW Interfaces

```
// Transfer software variables into RCU register
  rc[2] = j;
  rc[6] = k;
Loophead: // Destination jump label for restarting RCU after exception processing
  rc[4] = i;

  // Start RCU execution and wait for completion indicator (interrupt)
  rc[HW_START_REG] = 1;
  acev_wait();

  // OK, RCU execution stopped. Find out why ...
  if (rc[HW_EXIT_REG] == HW_EXIT_A) { // RCU indicated overflow of temporary value.

    // Fetch current values from RCU registers into software variables
    j = rc[2];
    i = rc[4];

    // Execute rest of this iteration in software
    printf("j=%d too large in loop i=%d\n", j, i);
    i = i + 1;

    // Now execute next iteration
    goto Loophead;
  } else /* HW_EXIT_B: RCU indicated normal exit */ {
    // Fetch final result from RCU register into corresponding variable
    j = rc[2];

    // Finish by executing remaining non-kernel instructions in software
    printf("result: j = %d\n", j);
  }
```

Debugging

- ❑ Should not be necessary with fully automatic tools
  - ❍ … but accidents happen, so:
- ❑ Allow single-stepping of hardware
  - ❍ Debug control block in hardware "wrapper"
- ❑ RCU registers holding variables are CPU-readable
  - ❍ Without need for external debug support
    - E.g., Xilinx ChipScope
  - ❍ Symbol tables associate register with variable names
    - Even more difficult than optimizing compilers
    - Consider speculative execution

Performance Optimization

- ❑ Example application was unspectacular
  - ❍ At best, 3 parallel operations (FSM: 2, 4, 7)
- ❑ Current compiler does not exploit, e.g.,
  - ❍ Dynamic hardware/software selection
  - ❍ Vectorization of array operations (SIMD)
  - ❍ Multi-threading (cache miss stalls entire datapath)
  - ➨ Much potential for achieving real speed-ups
    - Today: On GARP, 4x over MIPS on image compression
- ❑ Much lore from parallel / vector / VLIW compilers
  - ❍ Often applicable to hardware compilation
    - Huge suite of beneficial loop transformations

RCU-executable IP Blocks

- ❑ Despite best efforts:
  - ❍ Compilers are at best "good enough"
  - ❍ But cannot replace human expert
    - Assembly language programming
    - Highly optimized libraries for
      - Math, DSP, graphics, etc.
    - Easy interoperation with compiled code
      - Linking of object files
- ➨ Similar capability required for RCU compilers
  - ❍ But "linking" is more difficult
    - Much more freedom in hardware
      - Plethora of custom interfaces and data formats
      - Actually exploited for performance / area reasons
  - ❍ Shared resources must be managed (e.g., memory)
- ❑ Ongoing research

Practical Tips & Tricks

- ❑ For high-performance solutions
  - ❍ Don't just translate a software program
  - ❍ Think "hardware"
    - Digital signal processing started in late 1950's
      - *Without* software programmable processors
    - Everything realized in custom hardware
    - Many algorithms suited for RCUs buried in dusty tomes
- ❑ Examples
  - ❍ Coordinate Rotation Digital Computer (CORDIC)
    - Approach to calculate trigonometric and other transcendental function using just shifts and adds
  - ❍ Vector magnitude of (a,b)
    - Expensive: m = sqrt(a*a + b*b)
    - If 10% inaccuracy is OK: m' = max(a,b) + 0.5 min(a,b)

Custom Number Formats

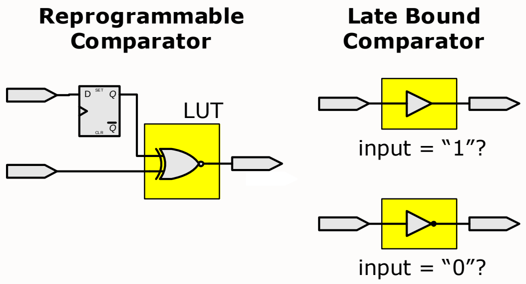
- Simple: Match operator width precisely to data
  - Only internally, external I/Os are fixed width
  - Example: 8b + 12b = 20b instead of 32b ops
- Medium: Modified standard formats
  - Custom fixed point formats: 8b.4b
  - Custom floating point formats
    - E.g., increased precision, reduced dynamic range
  - Match to requirements at specific points in algorithm
- Complex: Non-standard numerical representations
  - For Number Theoretic Transforms (can outperform FFT)
    - 1's Complement (Mersenne), Diminished 1 (Fermat)
- Good overview of techniques
  - Uwe Meyer-Baese
    - Digital Signal Processing with Field Programmable Gate Arrays, Springer 2001

Partial Evaluation

- Reduce hardware size by propagating known constants through circuit

- Occurs when creating circuit structure
  - Circuit synthesis for HDL-based design flows
  - Within parameterized module generators
- Very common use: constant coefficient multipliers
  - See previous HLL compilation example
- Other applications:
  - Encryption-key specific RCUs

Late Binding



- Limited form of run-time reconfiguration
  - Change circuit function
  - ... But retain structure of mapped circuit
    - Number and interconnection of logic elements constant
  - Only contents of logic elements are changed
- More area and delay efficient than reprogramming
- Value changes are often slower due to (partial) RTR

Multi-Bank Memories

- RCUs often have dedicated memory banks
  - On-chip memory blocks
  - External memories
  - In general fast SRAM
- Allows multiple simultaneous memory accesses
  - Can greatly improve throughput
- When programming for micro-processors
  - Homogeneous memory space
  - At best: Consider locality (cache characteristics)
- Using multi-bank memory system
  - Organization exposed to programmer
  - Data distribution across banks crucial

Off-the Shelf Technologies

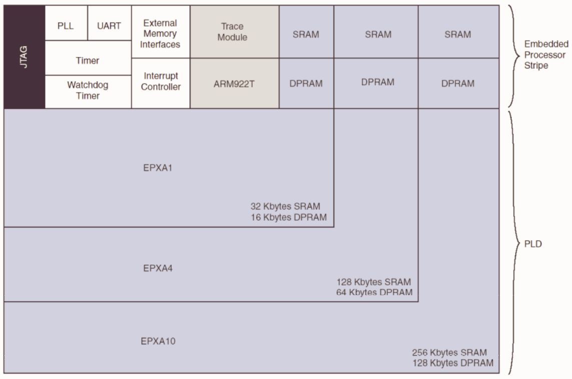
- System FPGAs
- Adaptive Computing Devices
- Reconfigurable IP Blocks

- (Configurable Processors)

System FPGAs

- Altera Excalibur
- Triscend A7/E5
- Xilinx Virtex II Pro

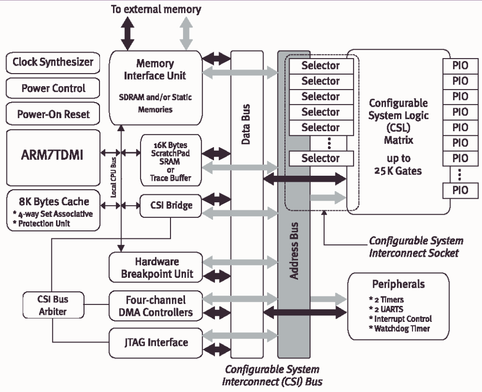
Altera Excalibur



- ARM922T core @ 200MHz
- Max ~ 1 M config gate capacity, up to 256 KB RAM
- DRAM memory controller (SDR and DDR)
- UART, IRQ controller, timer, watchdog, …

Figure from Altera Corp.

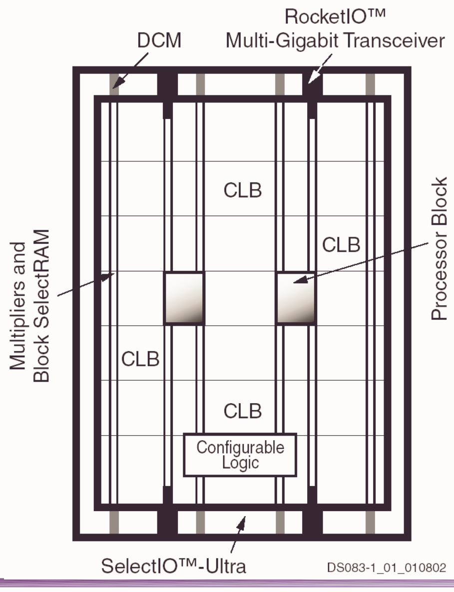
Triscend A7/E5



- A7: ARM7TDMI @ 60 MHz, E5: 8051 @ 40 MHz
- ~25 K configurable gate capacity
- 16 KB internal RAM
- DRAM memory controller (SDR and DDR)
- UART, IRQ controller, timer, watchdog, …

Figure from Triscend Corp.

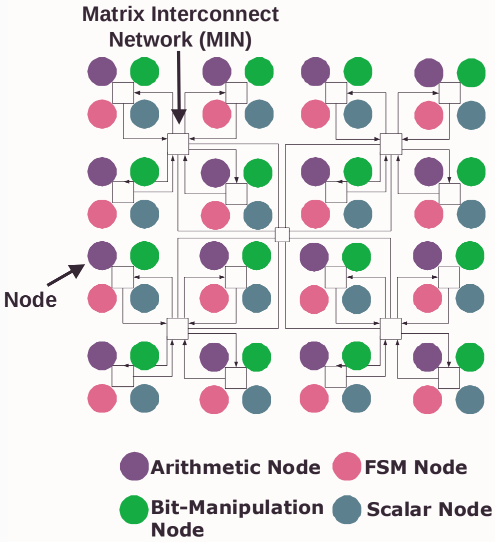
Xilinx Virtex II Pro



- 1-4x PPC405 @ 300+ MHz
- Max 4 M gates capacity
  - Up to 216 18bx18b multipliers
  - *No* hardwired interfaces/peripherals
- 486KB RAM

Figure from Xilinx Corp.

Quicksilver ACM



- Heterogeneous architecture
- Hierarchical (fractal) interconnection network
- Distributed memories
- Single cycle configuration

Courtesy QuickSilver Tech.

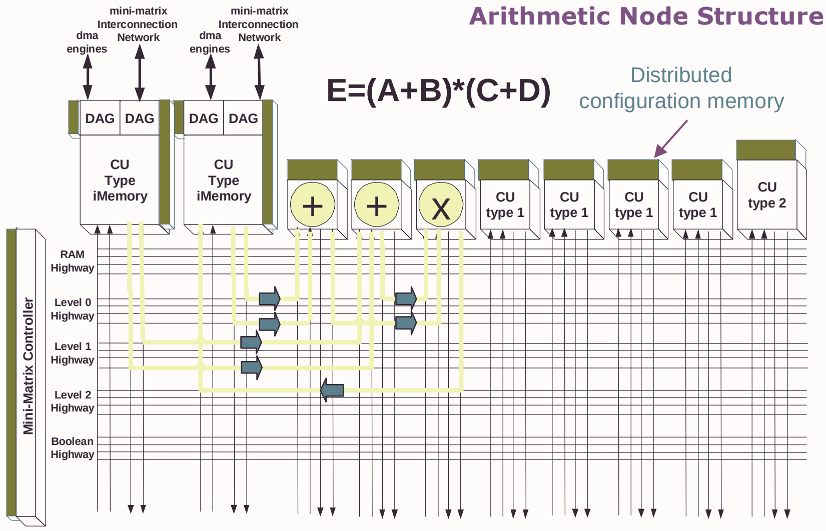
Quicksilver ACM cont'd.

- ❑ **Arithmetic node**
  - Implements different, linear, variable-width, arithmetic functions clock-cycle-by-clock-cycle
  - Implements different, non-linear, variable-width, arithmetic functions clock-cycle-by-clock-cycle
- ❑ **Bit-manipulation node**
  - Implements different, variable-width, bit-manipulation functions clock-cycle-by-clock-cycle
- ❑ **Finite state machine node**
  - Implements different, high-speed, complicated, finite-state machines clock-cycle-by-clock-cycle
- ❑ **Scalar node**
  - Implements different, complicated control sequences
- ❑ **Configurable input/output node**
  - Implements different interfaces to external interfaces such as buses

Courtesy QuickSilver Tech.

Quicksilver ACM cont'd.

Arithmetic Node Structure



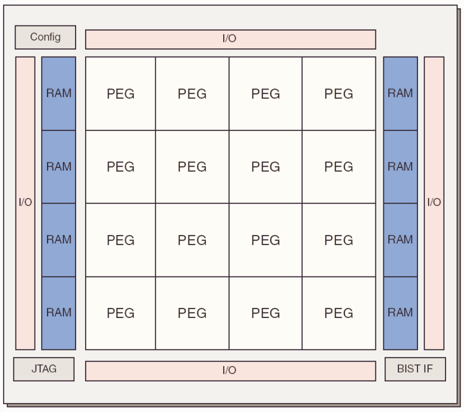
Courtesy QuickSilver Tech.

Reconfigurable IP Blocks

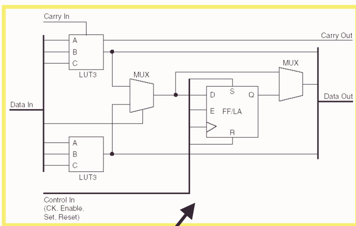
- ❑ Actel VariCore
- ❑ eASIC
- ❑ Elixent D-Fabrix
- ❑ IBM/Xilinx
- ❑ IP Flex
- ❑ Leopard Logic
- ❑ M2000 FLEXEOS
- ❑ PACT XPP
- ❑ picoChip

- ❑ Can be combined with configurable processor
  - ❍ Tensilica Xtensa
  - ❍ ARC ARCtangent
- ➨ Reconfigurable custom instructions

Actel VariCore



* 0.18um technology
  - CMOS SRAM
* Max. 250 MHz operation
* Uses 5 metal layers
* GDSII deliverable
* 5K - 40K ASIC gates



- ❑ Building blocks
  - ❍ PEG Blocks: 8x8 4x *Logic Unit*
  - ❍ RAM Blocks: 512x 18b RAMs
- ❑ Sizes: 2x1 … 4x4 PEGs, 0 … 8 RAMs

Figure from Actel Corp.

Actel VariCore cont'd.

❑ **Dual-pronged tool flow**

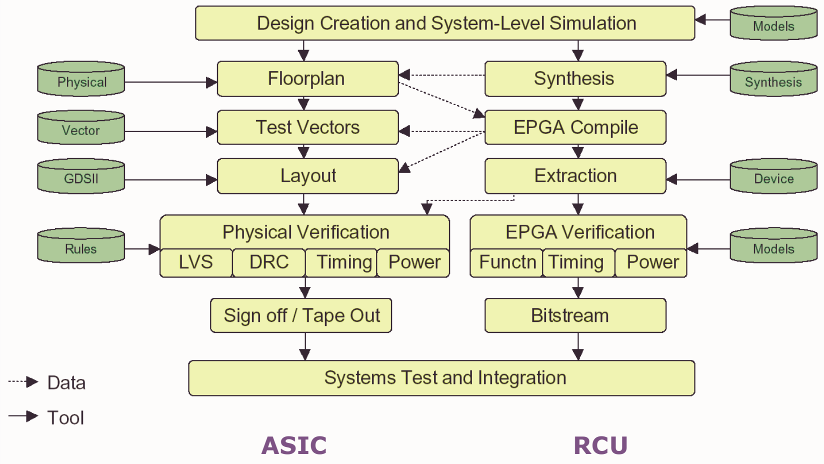


Figure from Actel Corp.

M2000 FLEXEOS

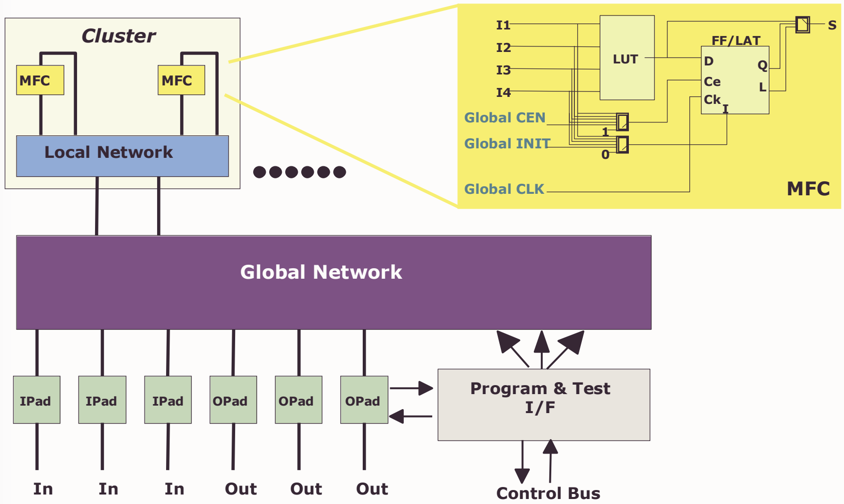
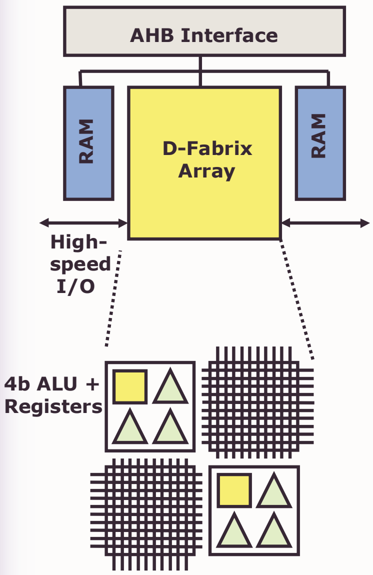


Figure from M2000 S.A.R.L

M2000 FLEXEOS cont'd.

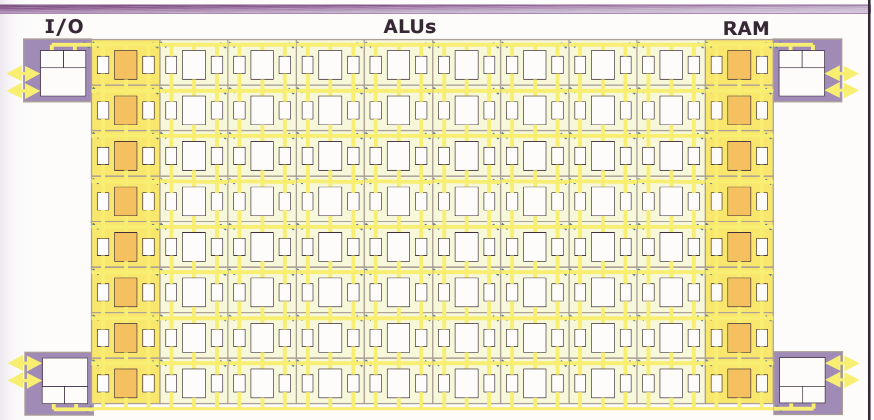
- Equivalent ASIC gate capacity up to 25 K gates
  - 200K FPGA equivalent gates
  - Example configuration: 3,000 MFCs
- Size of 8 sq. mm on ST HCMOS8, 0.18µm
- Programmability:
  - Configuration size 48KB
  - Loading time: <500 µs at 100MHz
    - Suitable for dynamic reconfiguration!
- Maximum measured frequency is 340MHz
  - Typical system clock 120MHz
- Very low power requirements:
  - Standby current less than 100µA
  - 100mW power consumption for 120 counters at 66MHz

Elixent D-Fabrix



- Based on HP Labs CHESS array
  - Max. 2048 ALUs, 256KB RAM
    - Other configurations possible
  - Fast reconfiguration
    - 32b,64b configuration ports
  - GDSII for CMOS SRAM
    - 0.18um
    - 0.13um
- Programmable in
  - Verilog, VHDL
  - Handle-C, MATLAB

PACT XPP



- Array of multi-bit ALUs
- Embedded RAM blocks
- High-speed interfaces for streaming I/O

Figure courtesy PACT XPP

PACT XPP cont'd.

- Delivered in RTL HDL as synthesizable soft-core
  - Targetable to 0.13um and 0.09um processes
- Parameters
  - Array size
  - ALU word width
  - Routing channels
  - RAM block size
- Wrapped in 1 … 2 external AHB interfaces
  - Connect to XPP-internal I/O streams
- Fast run-time-reconfiguration
  - 43b wide configuration bus
  - Multiple parallel configuration busses possible
  - 15us configuration time for 8x8 array

TI C6203 DSP ./. PACT XPU128

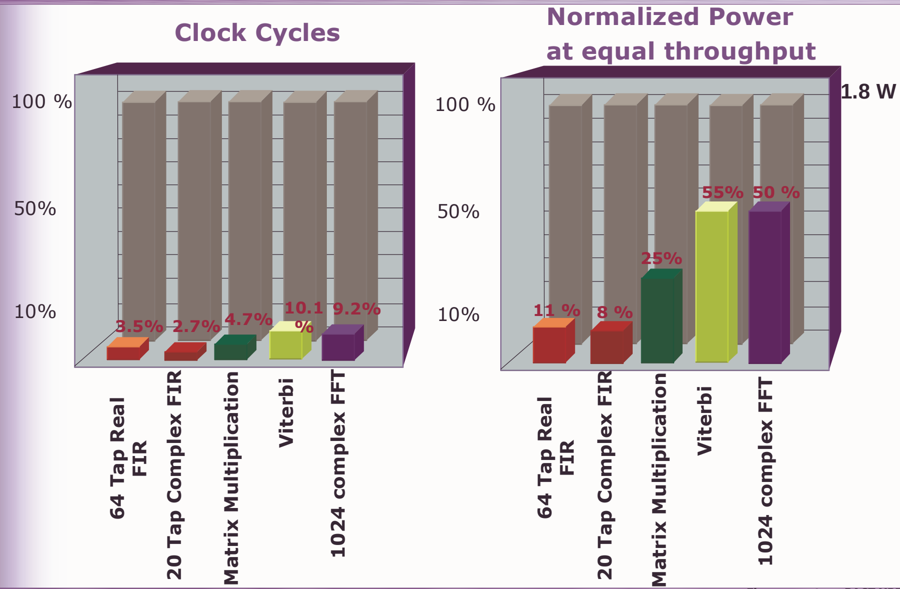


Figure courtesy PACT XPP

Conclusions

- Reconfigurable computing has much potential
  - Performance
  - Power
- Trends
  - Higher integration density
  - Exploitable dynamic reconfiguration
  - Tool support for higher-level programming
- Wide range of architectures
  - Match to specific application (domain)
- Most important recent development

Reconfigurable Systems-on-Chip

Color Slides in Soft-Copy

- http://www.cs.tu-bs.de/eis/koch/koch-date03.pdf