RaPiD
The Reconfigurable Pipelined Datapath

Presented and Slides Created by
Shannon Koh

Based on material from the University of Washington CSEís RaPiD Project (some images from website, talks and papers)

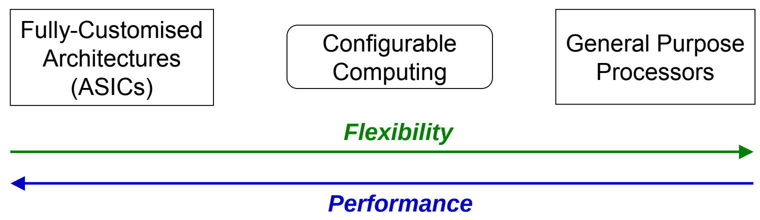
Overview

- **Motivation**
- Architecture Design
  - Datapath
  - Control
  - Memory
- Benchmark Architecture
- General Architecture
- RaPiD-C
- Compilation

Motivation

- Large data sets and computational requirements; e.g.
  - Motion Estimation for Real-Time Video Encoding
  - Accurate Low-Power Filtering for Wireless Comm.
- Target architectures include:
  - General Purpose Processors (including DSPs)
  - Application-Specific Integrated Circuits (ASICs)
  - Field-Programmable Compute Engines

Target Arch. Alternatives



- Target applications should have:
  - More computation than I/O
  - Fine-grained parallelism
  - Regular structure

General Purpose Processors

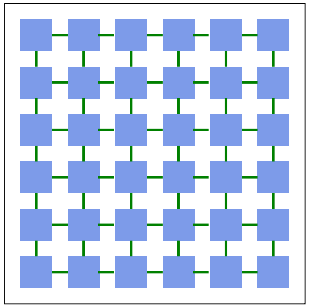
- Most flexible architectures
- Substantial die area allocated to:
  - Data and instruction caches
  - Crossbar interconnect of functional units
  - Speculative execution and branch prediction
- Can extract some instruction-level parallelism
- But not large amounts of fine-grained parallelism in compute-intensive applications

ASICs

- Higher performance (specific application, entirely inflexible)
- Lower cost, BUT:
  - High non-recurring engineering costs
- Speeds up only one application
- Only good for applications which are:
  - Well-defined
  - Wide-spread

Field-Programmable Computing

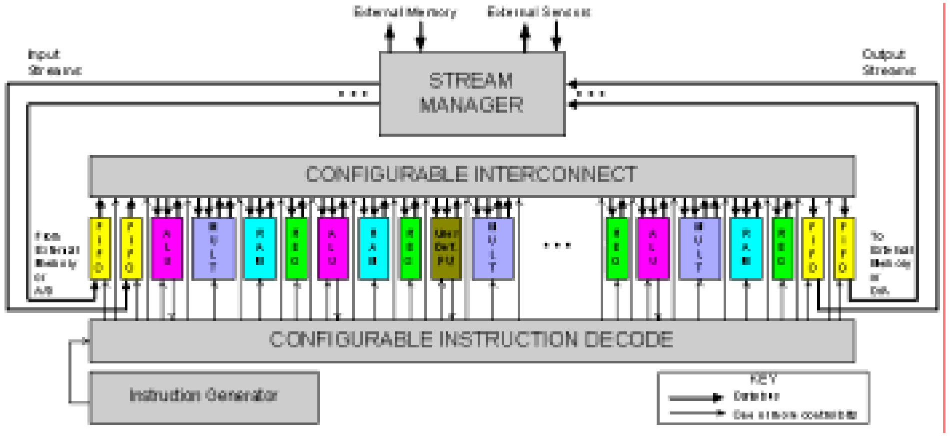
- Bridging flexibility and performance
- Reconfigurable to suit current application needs
- BUT, Implemented using FPGAs
  - Very fine-grained, therefore overhead due to generality is expensive (area and performance)
  - Programming FPGAs is either:
    - Poor in density or performance (using synthesis tools)
    - Requires intimate knowledge of the FPGA (manually)



The Solution?

- Given a restricted domain of computations, use *reconfiguration* to obtain a*:*
  - Cost advantage (one chip, many applications)
  - Performance advantage (customised to the domain)
- How?
  - Many customised functional units (hundreds)
  - Data cache → Directly streamed to/from external memory
  - Instruction cache → Configurable controllers
  - Global register file → Distributed registers/small RAMs
  - Crossbar interconnect → Segmented buses

RaPiD ñ Reconfigurable Pipelined Datapath



Pros and Cons

- **Pro**: Removal of caches, crossbars and register files frees up area that could be used for compute resources
- **Pro**: Communication delay is reduced by shortening wires
- Con: Reduces types of applications (e.g. highly irregular, little reuse, little fine-grained parallelism)
- **Pro**: Regular computation-intensive tasks like DSP, scientific, graphics and communications applications will be better over G.P. architectures, and is more flexible than an ASIC.

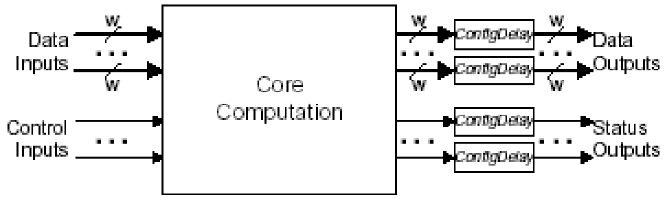
Overview

- Motivation ✓
- **Architecture Design**
  - **Datapath**
  - Control
  - Memory
- Benchmark Architecture
- General Architecture
- RaPiD-C
- Compilation

Datapath Architecture

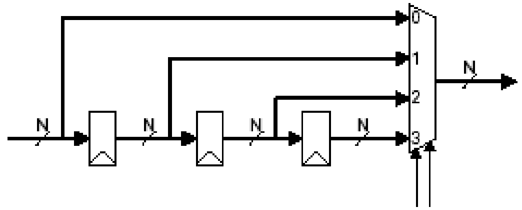
- Hundreds of functional units; broad complexity range
- Coarse-grained, word-based
- Linearly arranged with word-based buses
  - Simplifies layout and control
  - Tightly spaced, no corner turning switches
  - Multidimensional algorithms can be mapped
- Exceptions/control handled by tag bit in data value (propagated to future function units)

Functional Units



- Interconnect → Computation → Interconnect
- *ConfigDelay* allows for deeper pipelining
- Examples include ALUs, multipliers, shifters, memory, specific functions (e.g. LUTs, no control input), configurable functions (e.g. bit manipulations)

ConfigDelay Unit

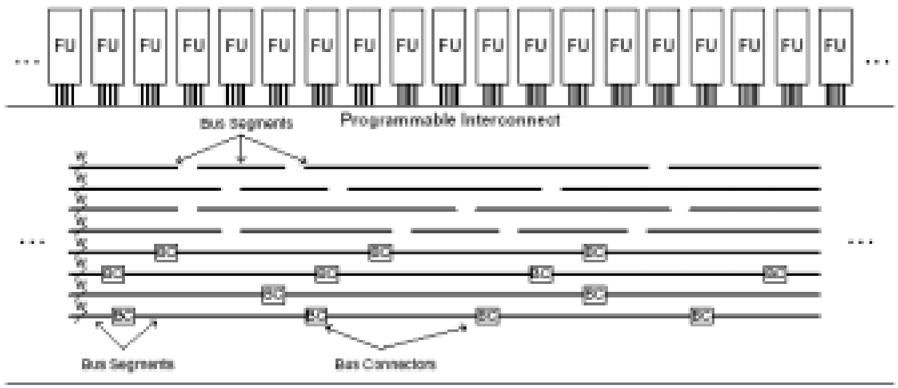


- Delay by up to three registers
- Deeper pipelining

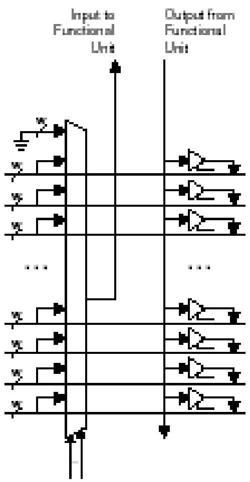
Configurable Interconnect

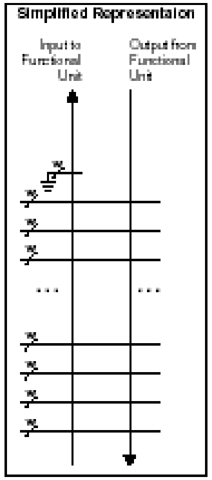
- A set of segmented tracks running the entire length of the datapath
- Segmented buses are connected with bus connectors
  - Left/Right/Both Driving
  - *ConfigDelay* included
- Double-width data can be output to two tracks and input to two functional units

Configurable Interconnect



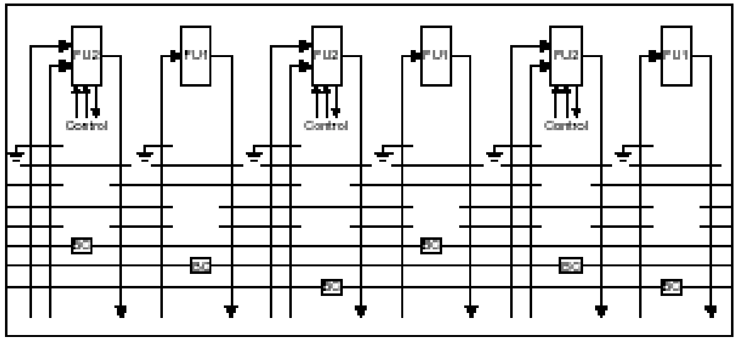
At the Functional Unit



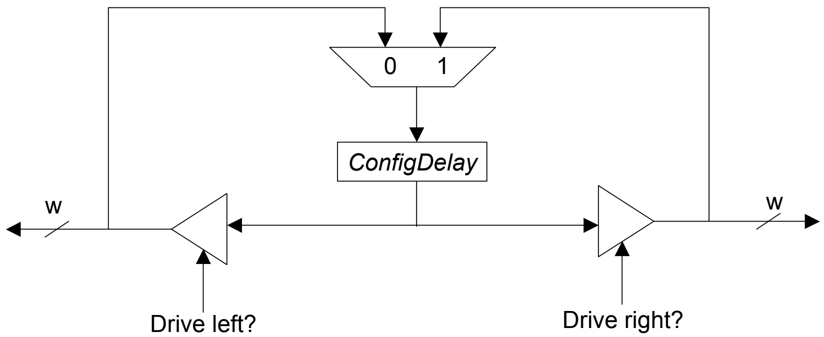


Cells

- Functional units are grouped to form a cell
- Cells are replicated to form the entire datapath



Bus Connector



Overview

- Motivation ✓
- **Architecture Design**
  - Datapath ✓
  - **Control**
  - Memory
- Benchmark Architecture
- General Architecture
- RaPiD-C
- Compilation

Control Architecture

- Control bits used in interconnect (multiplexer, tristate drivers, *ConfigDelay* and bus connectors) and function units
- Static field-programmable bits
  - Too inflexible ñ only good for static dataflow networks
- Programmed control
  - Too wide, therefore very expensive per cycle

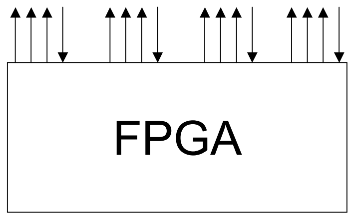
Control Architecture

Unit	Bits/ Unit	Units/ Cell	Bits/ Cell	Soft	Hard
Multiplexer	3	9	27	27	0
Tristate Driver	1	42	42	0	42
ConfigDelay	2	15	30	0	30
Bus Connector	2	6	12	0	12
FU1	0	3	0	0	0
FU2	2	3	6	6	0

Application Domain (Revisit)

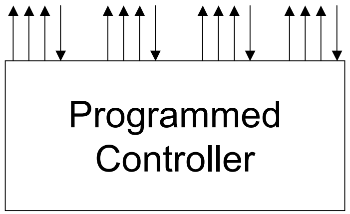
- Pipelined computations which are very repetitive
- Spend most of the time in deeply nested computation kernels
- Soft control is statically compiled
- How should we design the control architecture?

FPGA Control



- State machines mapped to an FPGA
- Not very efficient due to performance of FPGA
- But, easy to reconfigure

Programmed Control

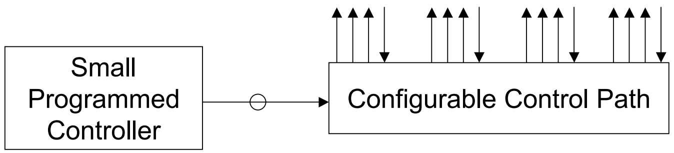


- Dedicated controller
- Better performance
- Less flexibility
- VLIW still expensive (area and performance*)

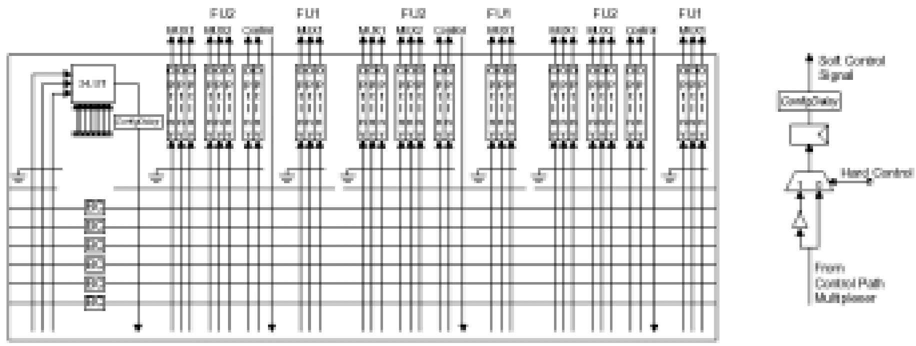
Reducing Instruction Length

- Most of the soft control is constant per application
- Regularity of computations allow much of the soft control to control more than one operation in more than one pipeline stage
- Reduce controller size
- Add a configurable control path

Controller and Decoder



Control Path Cell



Instruction Generator

```
for i = 0 to 9
   for j = 0 to 19
      for k = 0 to 29
         if (k == 0)
            load reg;
         if (j <= 3)
            inc ram addr;
         if (k > 5)
            w += w * y;
         if (k == 0 && j > 3)
            w = 0;
```

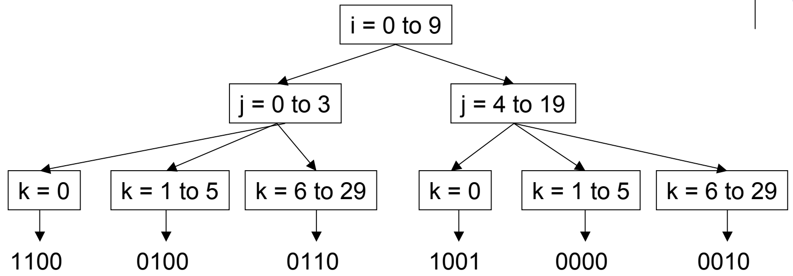
1xxx

x1xx

xx1x

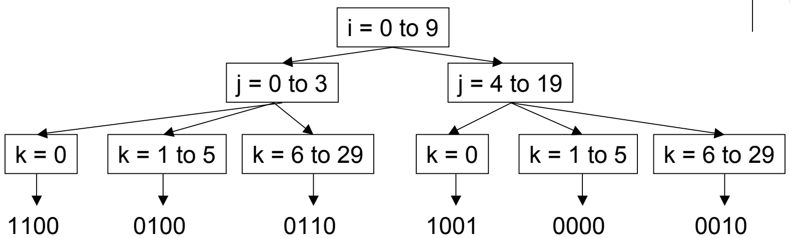
xxx1

Instruction Tree




```
for i = 0 to 9
  for j = 0 to 19
    for k = 0 to 29
      if (k == 0) 1xxx;
      if (j <= 3) x1xx;
      if (k > 5)  xx1x;
      if (k == 0 && j > 3) xxx1;
```

Instructions

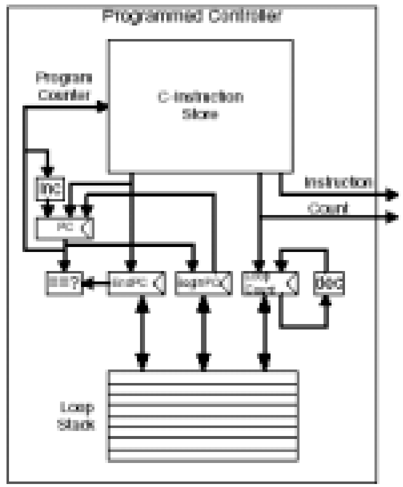


C-Instructions

```
      loop  10   end1
      loop   4   end2
      inst   1   1100
      inst   5   0100
end2: inst  24   0110
      loop  16   end1
      inst   1   1001
      inst   5   0000
end1: inst  24   0010
      halt
```

Instruction Controller

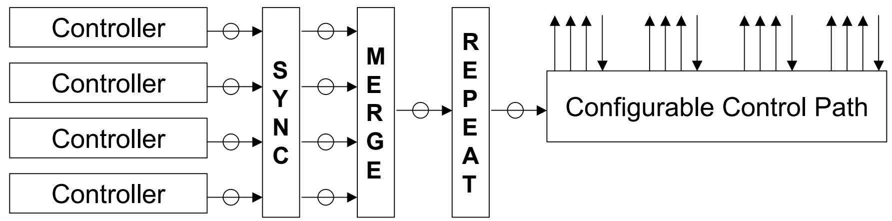
```
      loop  10   end1
      loop   4   end2
      inst   1   1100
      inst   5   0100
end2: inst  24   0110
      loop  16   end1
      inst   1   1001
      inst   5   0000
end1: inst  24   0010
      halt
```



Parallel Loop Nests

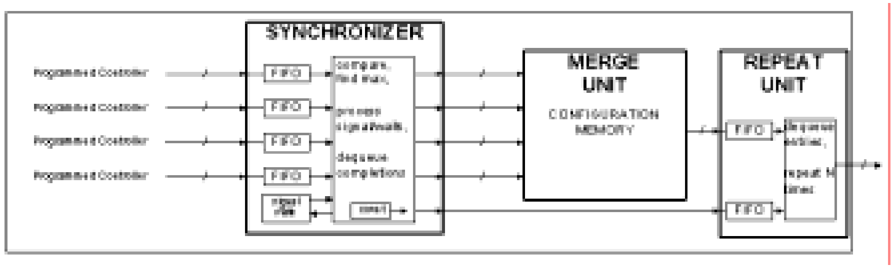
- Single controller ñ cross product of two loop nests to generate words (not good)
- Multiple controllers each executing one loop
- Synchronisation using primitives:
  - `signal NUM` : indicates that controller number ìNUMî should stop waiting or skip to its next wait if not waiting
  - `wait I` : repeats instruction word ì`I`î until a signal arrives

Parallel Loop Control



- Synchronisation handled by sync unit (*signal*, *wait*)
- Merge unit may be a bitwise OR or PLA if required
- Repeat unit handles repeat instruction repeats (*inst*)

Instruction Generators

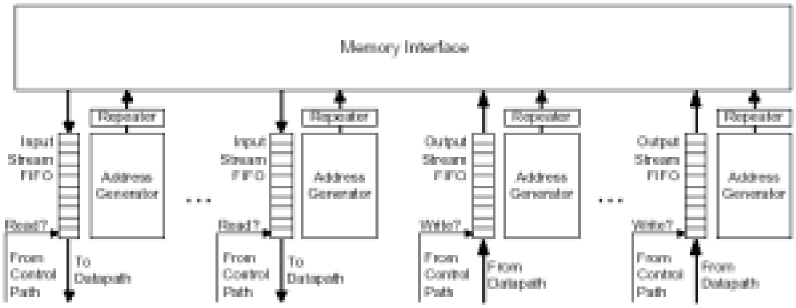


Overview

- Motivation ✓
- **Architecture Design**
  - Datapath ✓
  - Control ✓
  - **Memory**
- Benchmark Architecture
- General Architecture
- RaPiD-C
- Compilation

Memory Architecture

- Sequences of memory references are mapped to address generators
- Input FIFOs are filled from memory and output FIFOs are emptied to memory

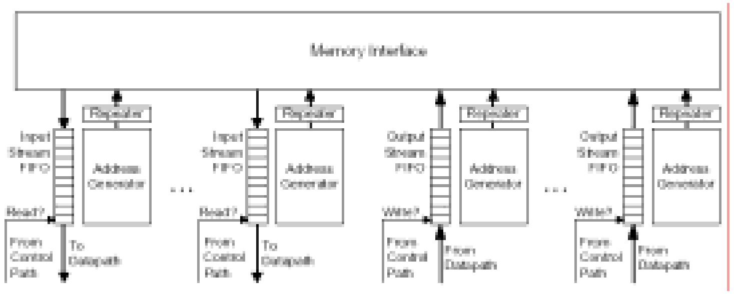


Memory Requirements

- Memory interface routes between streams and external memory modules
- High bandwidth through:
  - Fast SRAM
  - Aggressive interleaving and/or batching
  - Out-of-order handling of addresses
- Sustained data transfer of three words/cycle
- May also stream from external sensors

Address Generators

- Resembles programmed controller but produces addresses
- Addresses packaged with count and stride
- Repeaters increment addresses by the stride



Address Generators (Contíd)

- Addressing pattern statically determined at compile time
- Timing is determined by control bits
- Synchronisation achieved by halting the RaPiD array when:
  - FIFO is empty on a read
  - FIFO is full on a write

Overview

- Motivation ✓
- Architecture Design ✓
  - Datapath ✓
  - Control ✓
  - Memory ✓
- **Benchmark Architecture**
- General Architecture
- RaPiD-C
- Compilation

Benchmark Architecture

- Application domain consists primarily of signal-processing applications
- Requires high-precision multiply-accumulates
- 16-bit fixed-point datapath with 16×16 multipliers and 32-bit accumulates
- Cell comprises:
  - 3 ALUs and 3 64-word RAMs
  - 6 GP Registers and 1 multiplier

Benchmark (Continued)

- 14 data tracks
- 32 control tracks
- 16 replications of the cell
- Functional unit mix was chosen based on requirements of a range of signal processing applications

Characteristics

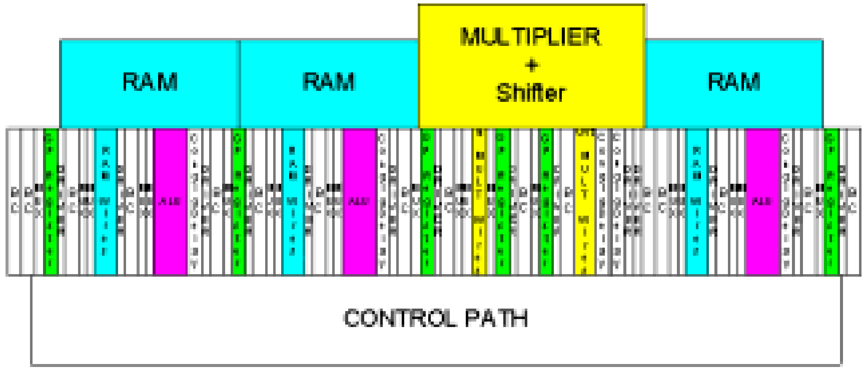
- .5μ process (λ = .3μ)
- 3.3v CMOS using MOSIS scalable submicron design rules
- 100 MHz clock
- 16-bit fixed point data, 16 Cells
  - 16 Multipliers
  - 48 ALUs
  - 48 RAMs (64-word)
- 14 data buses, 32 control buses

Area Requirements

Component	Area ($M\lambda^2$)	Number	Total Area ($M\lambda^2$)	% of cell Area
data memories	2.81	3	8.43	15.0%
multipliers	5.16	1	5.16	9.2%
ALUs	0.92	3	2.76	4.9%
general registers	0.39	6	2.32	4.1%
Functional Unit Subtotal			18.67	33.1%
Multiplier/RAM I/O routing			2.87	5.1%
Input multiplexers	0.22	20	4.41	7.9%
Output drivers	0.22	14	3.10	5.5%
Bus connectors	0.39	15	5.90	10.5%
Configurable delays	0.39	5	1.94	3.4%
Configurable Interconnect Subtotal			14.27	32.4%
Soft control bits	0.07	104	6.89	12.2%
Programmable logic blocks	0.35	3	1.06	1.9%
Bus connectors	0.01	104	1.44	2.6%
SRAM configuration cells	0.002	312	0.79	1.4%
Configuration memory overhead			2.72	4.8%
Control Subtotal			12.89	22.9%
Unused space			6.54	11.6%
Total cell area			56.35	100%

5.07 mm≤for λ = .3μ and 2.25 mm≤for λ = .2μ

FloorPlan



Configuration Overhead

- Straightforward interpretation: triples the area
- BUT, hardwired interconnect and control (e.g. FSMs) are called overhead here
- Hardwired circuits will not use all functional units or the full data width
- Configurable datapath evaluates a variety of computations
  - Approx. 67% RaPiD but 95-98% for FPGAs

Control Bits

Unit	Bits/Unit	Units/Cell	Bits/Cell	Soft	Hard
Multiplexer	4	20	80	80	0
Tristate Driver	1	196	196	0	196
ConfigDelay	2	26	52	0	52
Bus Connector	2	15	30	0	30
GP Register	0	6	0	0	0
ALU	7	3	21	18	3
RAM	3	3	9	6	3
Multiplier	8	1	8	0	8
Total			**396**	**104**	**292**

Power Consumption

- Optimised for performance rather than power
- But, features available for low power applications:
  - Turn off clocks to unused registers
  - Tie inputs of unused functional units to ground
- Thus, power only used for clocking used units and the clock blackbone

Application Performance

- Generally, 1.6 billion MACs per second
- FIR Filters
  - 16 tap, 100 MHz sample rate
  - 1024 tap, 1.5 MHz sample rate
  - 16-bit multiply, 32-bit accumulate
  - Symmetric filter, double performance
- IIR Filters
  - 48 taps at 33.3 MHz

Performance (Continued)

- Matrix Multiply
  - Unlimited size at 1.6 GMACS
- 2-D DCT (1.6 GMACS including reconfiguration)
- Motion Estimation
- Real-time Video
  - 12 fps for DCT+Motion on 720×576 image
    - Including 4000 reconfiguration cycles
  - 24 fps if double-gauged
- 2-D Convolution
- FFT possible but harder to program

Overview

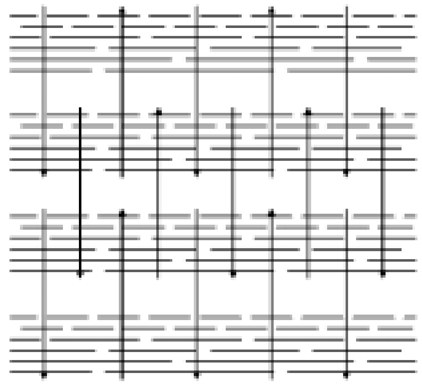
- Motivation ✓
- Architecture Design ✓
  - Datapath ✓
  - Control ✓
  - Memory ✓
- Benchmark Architecture ✓
- **General Architecture**
- RaPiD-C
- Compilation

General Architecture

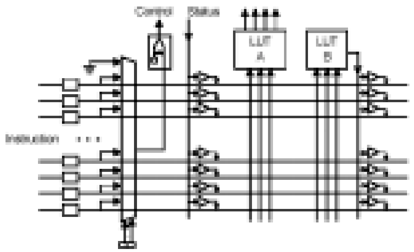
- Excludes/replaces implementation-specific components from the benchmark architecture
- Function units have an optional register for application-specific memory instead of *ConfigDelay*
- I/O ports are used instead of the memory interface

Generalised Interconnect

- Bypass paths take advantage of the 2-D nature of the layout
- Shortcut paths introduce hierarchy
  - Bus segments driven by other bus segments
- Allow dataflow graphs to be implemented more efficiently



Generalised Control

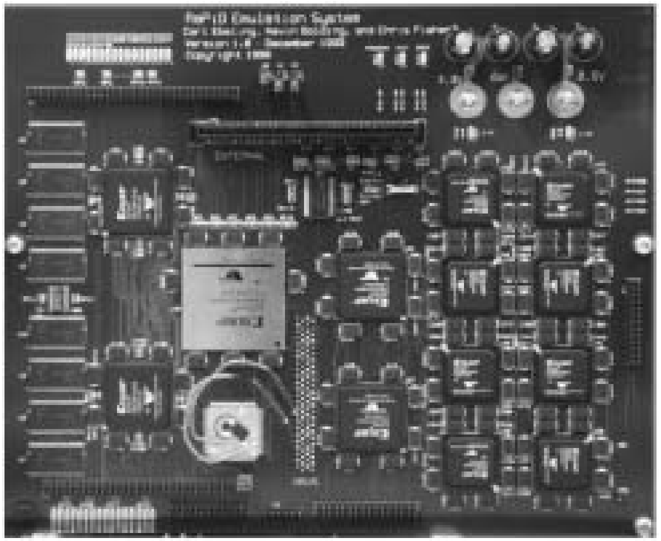


- Sequencer executes simple instructions such as branches, loops and subroutine calls with an instruction memory (perhaps RISC)
- LUTs can remap instruction bits to control bits, with additional status bits if necessary
- Benchmark architecture uses pipelined control (skewed control); general architecture allows control offset

RaPiD Emulator

- 13 Xilinx Virtex FPGAs
  - 9 for the datapath
  - 2 for a stream-based SDRAM memory subsystem
  - 1 for a stream switch
  - 1 for a control generator
- Sized to accommodate:
  - 32-cell RaPiD datapath
  - Typical cell size containing 1 multiplier, 3 ALUs, 3 memories and 6 datapath registers
- Interconnect is implemented per-application

RaPiD Emulator



Overview

- Motivation ✓
- Architecture Design ✓
  - Datapath ✓
  - Control ✓
  - Memory ✓
- Benchmark Architecture ✓
- General Architecture ✓
- **RaPiD-C**
- Compilation

RaPiD-C

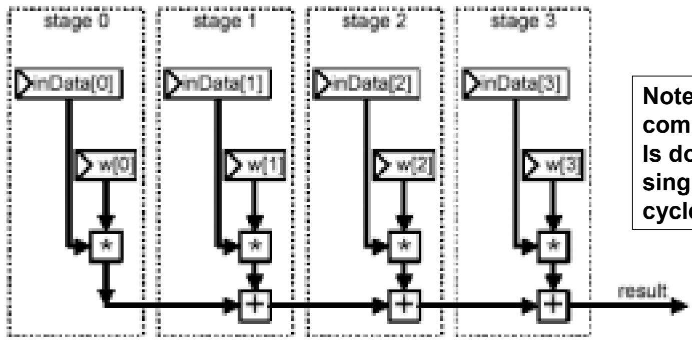
- Broadcast computation model
  - Compiler takes care of pipelining
- If STAGES pipeline stages were required, a loop like the following may be implemented:

```
for (s = 0; s < STAGES; s++)
```

- The `Datapath` instruction is ìshorthandî for the loop above e.g.

```
Datapath {
  if (s == 0) result = inData[s] * weight[s];
  else        result = result + inData[s] * weight[s];
}
```

Compiled Datapath



Note: Entire computation Is done in a single clock cycle.

```
Datapath {
  if (s == 0) result = inData[s] * weight[s];
  else        result = result + inData[s] * weight[s];
}
```

Control Loops

- Multiple cycle-loops

```
For i;

for (i = 0; i < N; i++) {
  Datapath {
    if (i == 0) result = inData[s] * weight[s];
    else        result = result + inData[s] * weight[s];
  }
}
```

- This does a MAC per clock cycle, therefore `i` is used to initialise the datapath at each cycle

Data and Variables

```
For i;

Word sum[STAGES], inData[STAGES], weight[STAGES];

for (i = 0; i < N; i++) {
  Datapath {
    temp[s] = inData[s] * weight[s];
    if (i == 0) sum[s] = temp[s];
    else        sum[s] = sum[s] * temp[s];
  }
}
```

- The compiler recognises that **`temp[s]`** does not require a register but **`sum[s]`** does

Streams

```
StreamIn strInput;
StreamOut strOutput’
Pipe inData;
For i, j, k;

Par {
thread:
for (i = 0; i < N; i++) {
  Datapath {
    if (s == 0) inData = strInput;
    if (s == STAGES – 1) strOutput = inData;
  }
}

vthread:
for (j = 0; j < N; j++) {
  Datapath {
    strInput = X[j];
  }
}
```

Streams (Continued)

```
vthread:
for (k = 0; k < N; k++) {
  Datapath {
    strOutput = Y[k]; // Note: Stream is the target of assignment
  }
}
}
```

- X and Y are external memory streams
- Stream definition and access must be decoupled
- Decoupling requires definition and access to be in parallel threads
  - Real threads: Maximum 4

Pipes

- Consider writing x[s] in stage s
- It cannot be read in stage s-1 or s+1

```
x[s + 1] = x[s]; // illegal
```

- Pipe variables must be used

Ideal Execution

```
Word inData[STAGES], weight[STAGES];
For i;
Word result[STAGES];

for (i = 0; i < N; i++) {
  Datapath {
    if (s > 0) result[s] = result[s – 1];
    // Initialisation
    if (s == 0)
      result[s] = inData[s] * weight[s];
    else
      result[s] = result[s] + inData[s] * weight[s];
  }
}
```

Pipe Variables

```
Word inData[STAGES], weight[STAGES];
For i;
Pipe result;

for (i = 0; i < N; i++) {
  Datapath {
    if (s == 0)
      result = inData[s] * weight[s];
    else
      result = result + inData[s] * weight[s];
  }
}
```

Unchanged Ripple

```
Pipe inData;

For i;

for (i = 0; i < N; i++) {
  Datapath {
    if (s == 0) inData = X[i];
    if (s == STAGES – 1) Y[i] = inData;
  }
}
```

More RaPiD-C

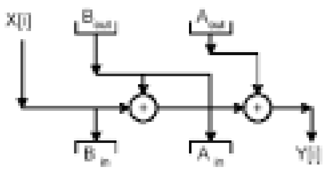
- RaPiD-C also supports
  - RAMs as basic building blocks
  - Pipe delays
  - No-Control (Always-Perform)
  - BackPipes
  - Generic Function Units
    - C Operators are compiled to function unit references
    - Custom combinational and sequential units can be specified

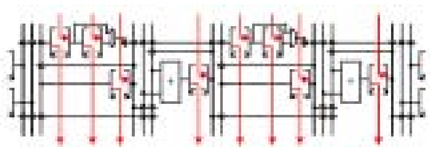
Overview

- Motivation ✓
- Architecture Design ✓
  - Datapath ✓
  - Control ✓
  - Memory ✓
- Benchmark Architecture ✓
- General Architecture ✓
- RaPiD-C ✓
- **Compilation**

Compilation

- Current work by the UW RaPiD team
- RaPiD-C is compiled to dataflow graphs
- Datapath graphs are created for RaPiD
- Datapath graphs then need to be scheduled over clock cycles





Other Requirements

- Time multiplexing (reconfiguration to support larger datapaths)
- Mapping to function units and memories
- Predicated execution (avoiding control hazards)
- Stitching graphs
- Scheduling for optimised control

Conclusions

- High performance, low cost for a good domain of computations
  - Due to *reconfiguration* to obtain a cost/performance advantage
- RaPiD can be low-powered
  - Local communication, distributed memories, clock disabling
- RaPiD can be a closely-coupled co-processor
- It can be an embedded system

References

- University of Washington CSE **RaPiD Project**
  - http://www.cs.washington.edu/research/projects/lis/www/rapid/
- Darren C. Cronquist, Paul Franklin, Chris Fisher, Miguel Figueroa, and Carl Ebeling. "**Architecture Design of Reconfigurable Pipelined Datapaths**," *Twentieth Anniversary Conference on Advanced Research in VLSI 1999*
- Carl Ebeling. ì**Rapid-C Manual**,î University of Washington Technical Report: UW-CSE-02-07-06, 2002.

More References

- C. Fisher, K. Rennie, G. Xing, S. Berg, K. Bolding, J. Naegle, D. Parshall, D. Portnov, A. Sulejmanpasic, and C. Ebeling. ì**An Emulator for Exploring RaPiD Configurable Computing Architectures**,î In *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications* (FPL 2001), Belfast, pp. 17-26, August, 2001.
- Carl Ebeling. ì**Compiling to Coarse-Grained Adaptable Architectures**,î University of Washington Technical Report: UW-CSE-02-06-01, 2002.

More References

- Carl Ebeling. ì**The General Rapid Architecture Description**,î University of Washington Technical Report: UW-CSE-02-06-02, 2002.
- ì**Architecture Design of Reconfigurable Pipelined Datapaths**,î talk presented at the Conference on Advanced Research in VLSI, Atlanta, March 1999.
  - http://www.cs.washington.edu/research/projects/lis/www/rapid/arvlsi_print/