Custom Processor Core Construction from C Code

Jelena Trajkovic and Daniel D. Gajski
Center for Embedded Computer Systems
UC Irvine

Outline

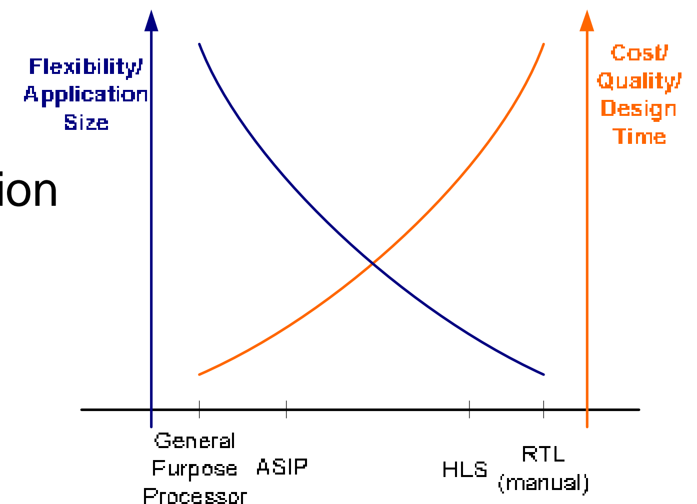
- Introduction
- Initial Data path Extraction
- Data path Optimization
- Experimental results
- Conclusion and Future Directions

Copyright © CECS 2008

Introduction

- Application implementation options

1. Chose processor, compile application
   - Cheaper, shorter time-to-market
   - Scalable with code size
   - May not have resources for optimal execution



2. Design HW for the application
   - Optimal components and structures
   - Longer development time
   - Not scalable

- Our goal: benefits from 1 & quality of 2

J. Trajkovic and D. Gajski
Copyright © CECS 2008

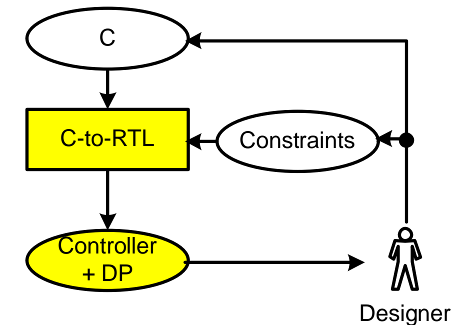
Problem Definition

- What is the best data path to execute given C code?
  - "The best" architecture is one that meets given constraints
  - Manual design
    - Not scalable for large application
  - Solution: Automatically extract data path from C code

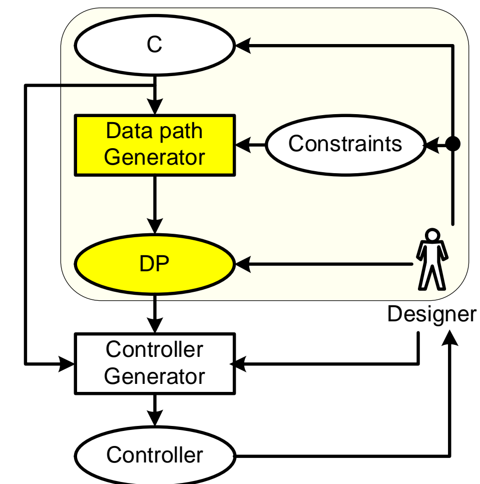
Copyright © CECS 2008

Approach (1/2)

- Traditional
  - Data path and controller are generated simultaneously
    - Applicable to small code size



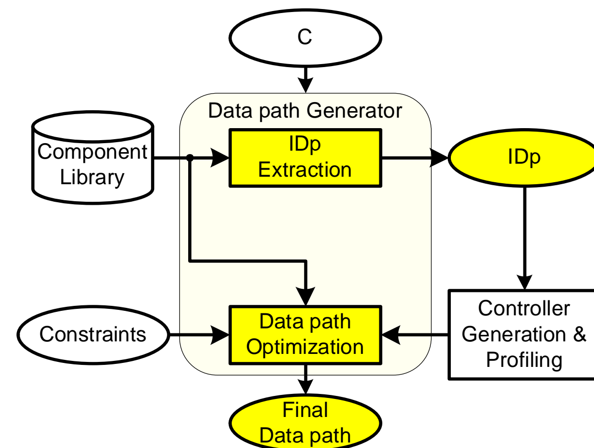
- Proposed
  - Separate *Data path* and controller generation
  - Derive Data path from C code



Copyright © CECS 2008

Approach (2/2)

- Our design flow
  - Extract *Initial Data path (IDp)*
    - For max performance
  - Optimize Data path
    - Iteratively based on designer constraints
- Benefits
  - Automate data path extraction
  - Standard input – C reference code
  - Scalability (~10000 lines of code)
  - Controllability of data path design
  - Quality



Copyright © CECS 2008

Outline

- Introduction
- Initial Data path Extraction
- Data path Optimization
- Experimental results
- Conclusion and Future Directions

Copyright © CECS 2008

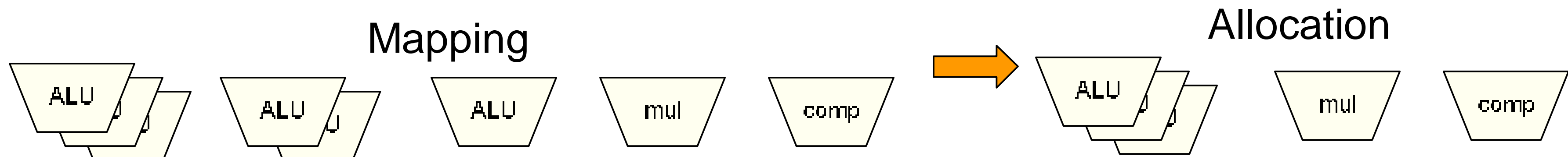
Initial Data path Extraction

- Set of C code properties
  - Data types, operators, parallelism, loops, dependences
- Set of HW components and templates
  - Functional units, storage elements, interconnect
  - Connectivity scheme, pipelining
- Matching heuristics
  - Date types and representation → bit-width
  - Parallelism → number of FUs, number of registers in RF, pipelining

Copyright © CECS 2008

Example: C Code Properties → HW

- Set of operations
  {+, -, shr, *, >}
- Max concurrent occurrence
  $m_+ = 3$, $m_- = 2$, $m_{shr} = 1$, $m_* = 2$, $m_> = 1$
- Mapping operations → FU
  + → ALU, - → ALU, shr → ALU, * → mul, > → comp



- Max concurrent transfers of source and destination operands
  - Source and destination busses
  - Greedy interconnect
- Create Initial Data path (IDp)

Copyright © CECS 2008

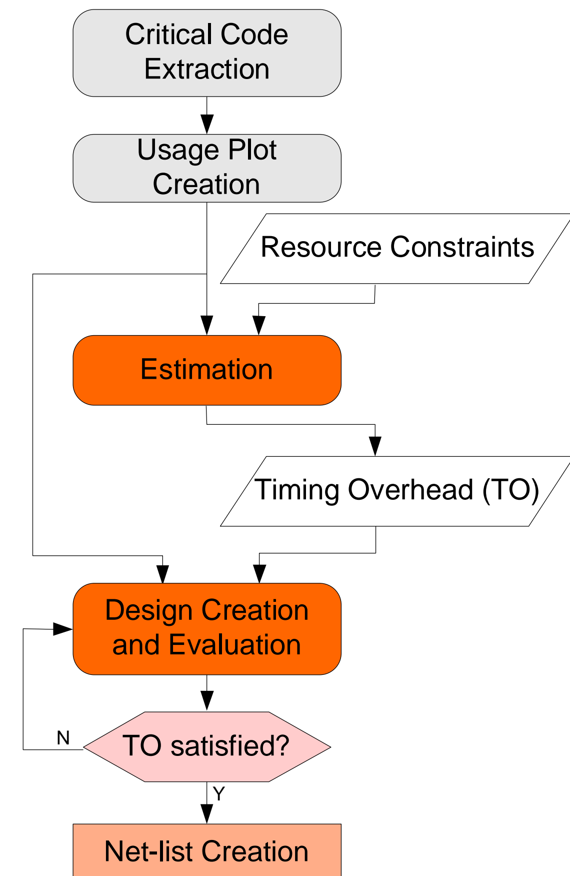
Outline

- Introduction
- Initial Data path Extraction
- Data path Optimization
- Experimental results
- Conclusion and Future Directions

Copyright © CECS 2008

Main steps in Data path Optimization

- Compile and profile IDp
- Select *Critical code* to be optimized
- Create *Usage Plot* for components
- Estimate *Timing Overhead* from *Resource constraints*
- Balance out the remaining components of Dp



Copyright © CECS 2008

Critical Code Extraction

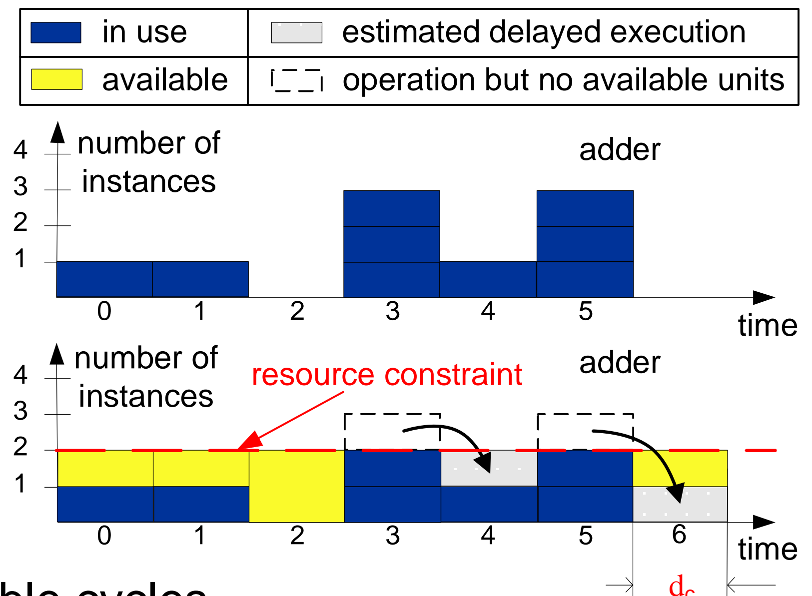
- Critical Code
  - Set of Basic Blocs (BB)
  - Contributes the most to the execution time

- Selects based on relative length (l) and frequency (f)
  - High length
  - High frequency
  - High frequency-length product
- Designer specifies Critical Code
  - By selecting parameters P_l, P_f, P_{fl}

Copyright © CECS 2008

Usage Plot and Timing Overhead

- For selected BBs create usage plot
  - Usage per cycle
  - Annotate with execution frequency

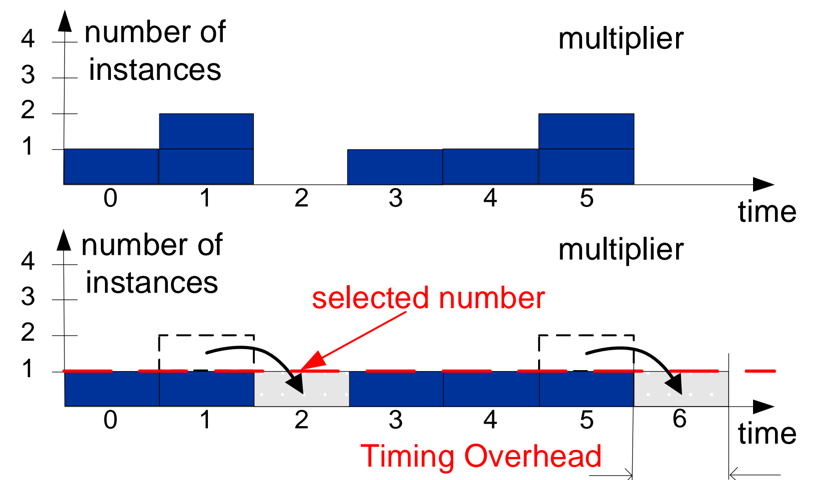


- Resource constraint
  - e.g. adder = 2
- Model delayed execution
  - Extra operations "executed" in available cycles
  - Estimate d_c – number of extra cycles
- The smallest d_c becomes *Timing Overhead*

Copyright © CECS 2008

Balancing Unrestricted Components

- Select designs to be evaluated
  - Set number of components with resource constraints
  - Set other resources to values in IDp
  - Select a component type to be varied



- E.g. multiplier: two designs
  - With one or with two multipliers
  - Having one multiplier satisfies

  Timing Overhead

Copyright © CECS 2008

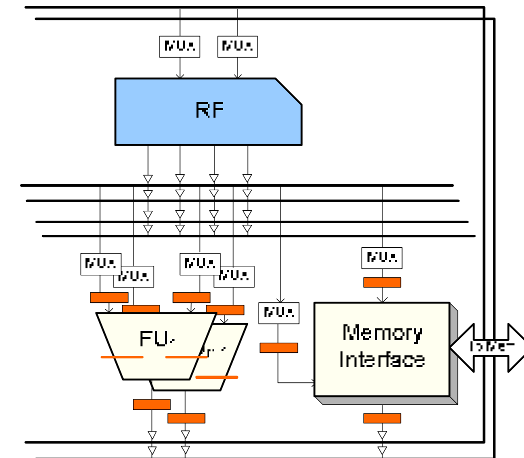
Outline

- Introduction
- Initial Data path Extraction
- Data path Optimization
- Experimental results
- Conclusion and Future Directions

Copyright © CECS 2008

Experimental Results

Bench	(P_l, P_f, P_{fl}) [%]	LoC	Gen. Time [sec]	
			Non-pipe	Pipe
bdist2	(60, 50, 45)	61	0.2	0.8
Sort	(80, 60,45)	33	0.1	0.1
dct32	(18, 65, 50)	1006	1.3	2.3
Mp3	(30, 55, 50)	13898	15.6	42.6



- Two set of experiments to be shown
  - Interactive design exploration
  - Design refinement quality
- Benchmarks
  - bdist2 (from MPEG2 encoder), Sort (bubble sort), dct32 (from MP3 decoder) and Mp3 (decoder)

Copyright © CECS 2008

Interactive Design Exploration

- Designer specifies
  - Partial resource constraint
  - Parameters for critical code extraction
  - Design choice (pipelined/non-pipelined)

- Baseline:
  - MIPS-style: ALU, multiplier, 128-entry RF2x1 and 2 source and 1 destination bus
  - A divider for Mp3 baseline implementation
  - Memory size customized per application

J. Trajkovic and D. Gajski
Copyright © CECS 2008

Description of Generated Architectures

- Difference from baseline

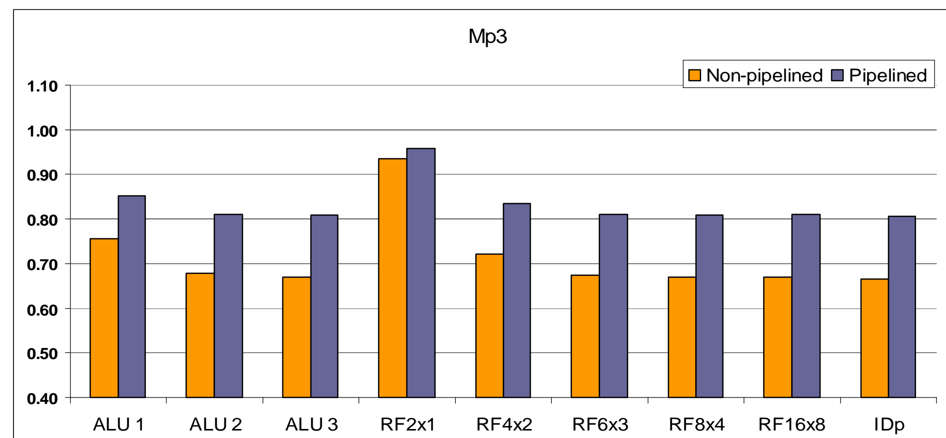
Bench	Pipe	ALU1	ALU2	ALU3	RF2x1	RF4x2	RF6x3	RF8x4	RF16x8	IDp
Bdist2	N	#R=64	#R=64	#R=64	#R=64	#R=64	#R=64	#R=64	#R=64	Rf 16x8, 3 Alu, 2 Mul
	Y	#R=32	#R=32	#R=32	#R=32	#R=32	#R=32	#R=32	#R=32	
Sort	N	#R=16	#R=16	#R=16	#R=16	#R=16	#R=16	#R=16	#R=16	Rf 16x8, 2 Alu
	Y	#R=16	#R=16	#R=16	#R=16	#R=16	#R=16	#R=16	#R=16	
dct32	N	Rf4x2	Rf6x3	Rf8x4	-	2 Alu	3 Alu	3 Alu	3 Alu	Rf 16x8, 3 Alu, 2 Mul
	Y	Rf4x2	Rf4x2	Rf6x3	-	2 Alu	2 Alu	2 Alu	3Alu	
Mp3	N	-	Rf6x3	Rf8x4	-	2 Alu	3 Alu	3 Alu	3 Alu	Rf 16x8, 3 Alu, 2 Mul
	Y	Rf4x2	Rf6x3	Rf6x3	-	2 Alu	2 Alu	2 Alu	2 Alu	

Copyright © CECS 2008

Example:Mp3

- Baseline
  - MIPS-style: ALU, multiplier, 128-entry RF2x1 and 2 source and 1 destination bus, divider

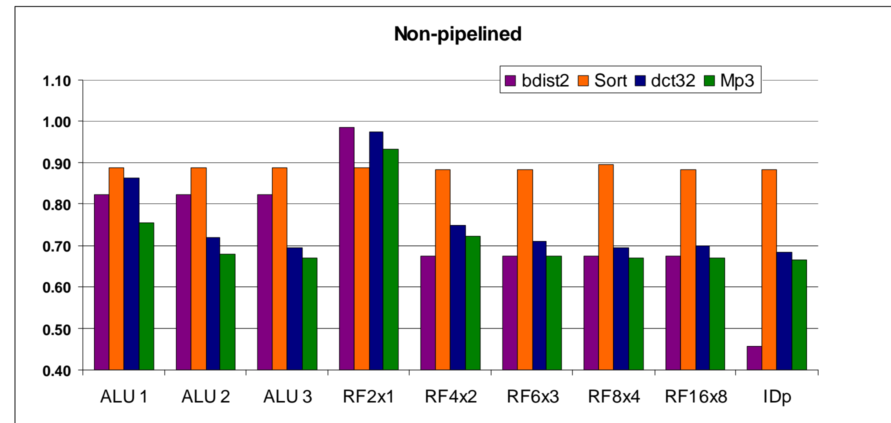
Bench	Pipe	ALU1	ALU2	ALU3	RF2x1	RF4x2	RF6x3	RF8x4	RF16x8	IDp
Mp3	N	-	Rf6x3	Rf8x4	-	2 Alu	3 Alu	3 Alu	3 Alu	Rf 16x8, 3 Alu, 2 Mul
	Y	Rf4x2	Rf6x3	Rf6x3	-	2 Alu	2 Alu	2 Alu	2 Alu	



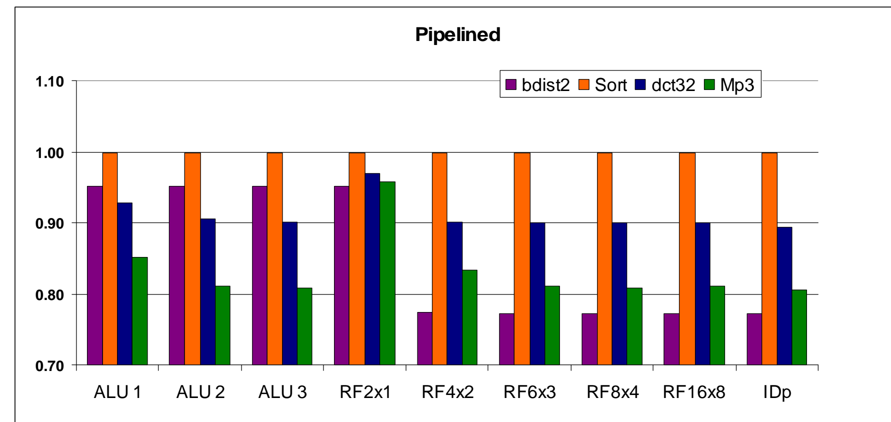
Copyright © CECS 2008

Number of Execution Cycles

- Non-pipelined
  - bdist2 - biggest saving for IDp
  - Sort – smallest improvement
  - dct32 – ALU3
  - Mp3 – ALU2 optimum



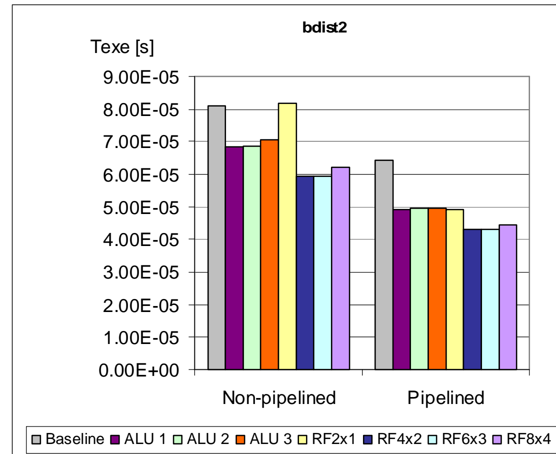
- Pipelined
  - less overall saving

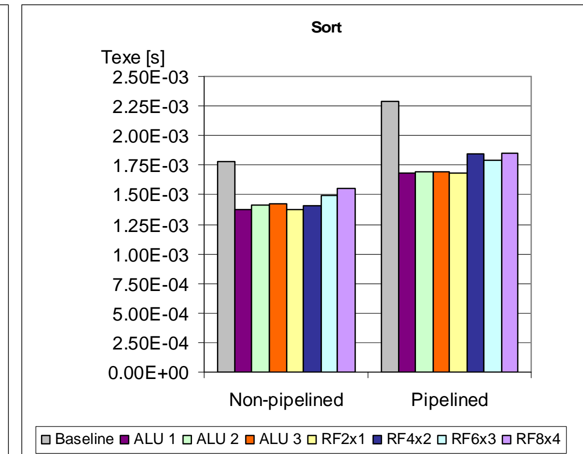


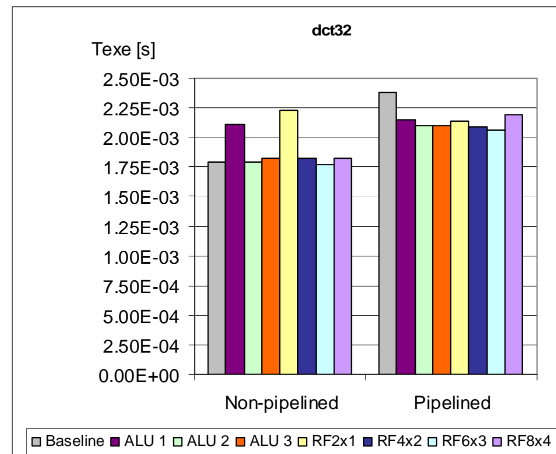
Copyright © CECS 2008

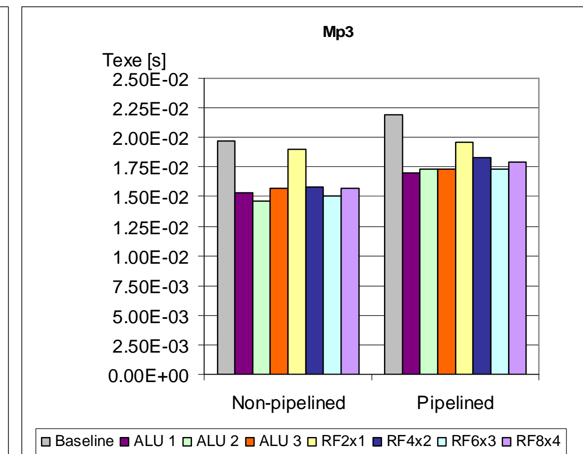
Total Execution Time

- Post-synthesis
- bdist2
  - pipelined RF4x2
- Sort
  - Non-pipelined ALU1
    - Reduced resources
- dct32
  - Non-pipelined ALU2
    - Large Tclk
- Mp3
  - Non-pipelined ALU2
    - Large Tclk





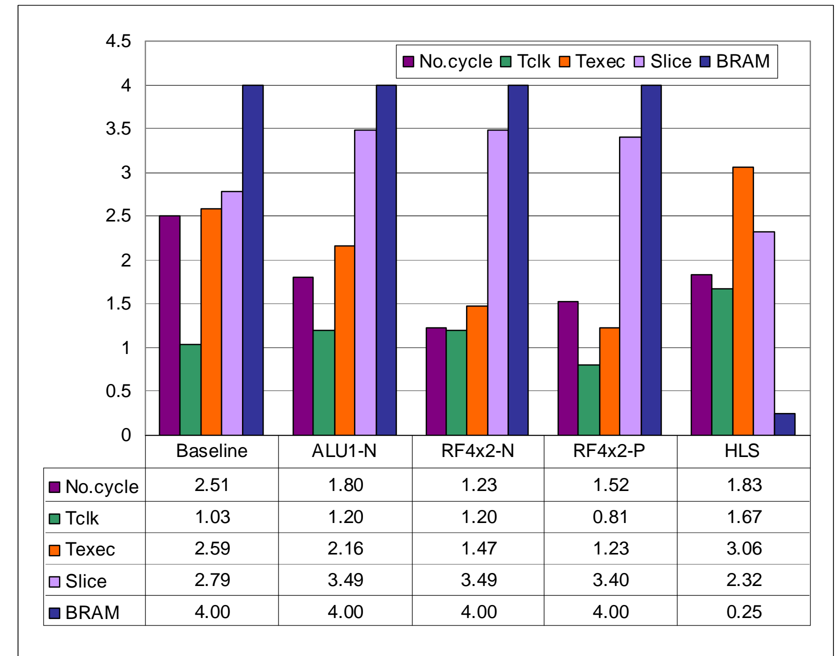




Copyright © CECS 2008

Design Refinement Quality

- Relative to manual design
  - dct32
- Generated vs. manual
  - No.cycles:
    - 23% - 80% extra
  - Tclk:
    - 29% speedup – 20% slowdown
  - Best Texe:
    - RF4x2: 23% extra
- Generated vs. academic HLS
  - Generated has better performance
  - HLS has better area



	Baseline	ALU1-N	RF4x2-N	RF4x2-P	HLS
No.cycle	2.51	1.80	1.23	1.52	1.83
Tclk	1.03	1.20	1.20	0.81	1.67
Texec	2.59	2.16	1.47	1.23	3.06
Slice	2.79	3.49	3.49	3.40	2.32
BRAM	4.00	4.00	4.00	4.00	0.25

- Design time: Generated → 2.3 sec vs. Manual → 3-man week

Copyright © CECS 2008

Conclusion

- Contributions
  - Algorithm for basic block analysis and core generation
  - Iterative algorithm for optimizing resource utilization
  - Demonstration on large C examples (>13K LOC)
- Benefits
  - Automatic C input to core datapath construction
  - Scalable to any size of C code
  - Datapath construction controllable using designer constraints
  - Generated core quality comparable to manual design
- Future work
  - Automatic pipeline configuration from C code
  - Forwarding based on static C code analysis

Copyright © CECS 2008

Acknowledgments

- The authors wish to thank Pramod Chandraiah for providing the Mp3 source code
- Many thanks to Pramod Chandraiah, Weiwei Chen, Gunar Schirner, Lin Yang and Bin Zhang for manual designs and Roger Ang, Hansu Cho and Dongwan Shin for manual and HLS designs for *dct32*
- We would also like to thank Mehrdad Reshadi and Bita Gorjiara for compiler support and Verilog generator

Copyright © CECS 2008

Thank You

- Questions?

Copyright © CECS 2008