



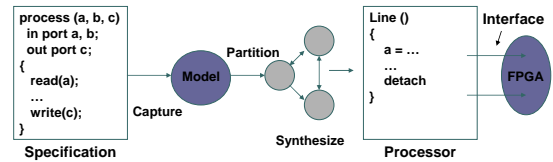
# CprE / ComS 583 Reconfigurable Computing

Prof. Joseph Zambreno  
Department of Electrical and Computer Engineering  
Iowa State University

Lecture #22 – Multi-Context FPGAs



## Recap – HW/SW Partitioning



- Good partitioning mechanism:
  - 1) Minimize communication across bus
  - 2) Allows parallelism → both hardware (FPGA) and processor operating concurrently
  - 3) Near peak processor utilization at all times (performing useful work)

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.2



## Recap – Communication and Control

- Need to signal between CPU and accelerator
  - Data ready
  - Complete
- Implementations:
  - Shared memory
  - FIFO
  - Handshake
- If computation time is very predictable, a simpler communication scheme may be possible

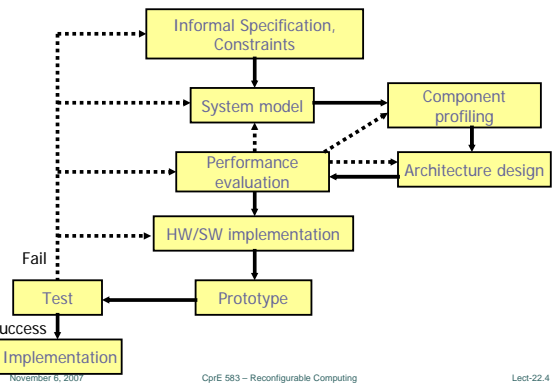
November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.3



## Recap – System-Level Methodology



November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.4



## Outline

- **Recap**
- **Multicontext**
  - Motivation
  - Cost analysis
  - Hardware support
  - Examples

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.5



## Single Context

- When we have
  - Cycles and no data parallelism
  - Low throughput, unstructured tasks
  - Dissimilar data dependent tasks
- Active resources sit idle most of the time
  - Waste of resources
- Why?

November 6, 2007

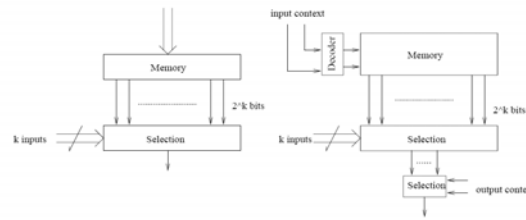
CprE 583 – Reconfigurable Computing

Lect-22.6

### Single Context: Why?

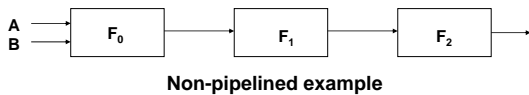
- Cannot reuse resources to perform **different** functions, only the **same** functions

### Multiple-Context LUT



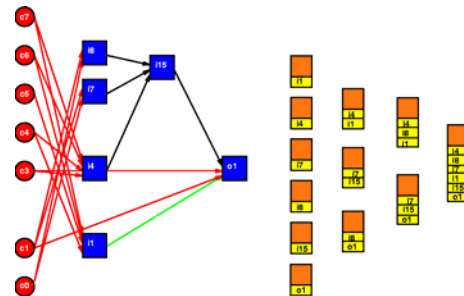
- Configuration selects operation of computation unit
- Context identifier changes over time to allow change in functionality
- DPGA – Dynamically Programmable Gate Array

### Computations that Benefit

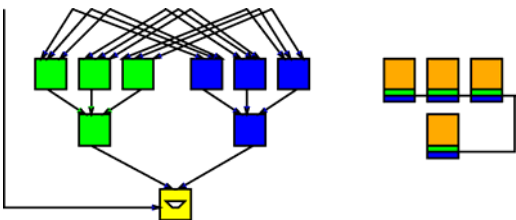


- Low throughput tasks
- Data dependent operations
  - Effective if not all resources active simultaneously
  - Possible to time-multiplex both logic and routing resources

### Computations that Benefit (cont.)



### Computations that Benefit (cont.)



### Resource Reuse

- Resources must be directed to do different things at different times through instructions
- Different local configurations can be thought of as instructions
- Minimizing the number and size of instructions a key to successfully achieving efficient design
- What are the implications for the hardware?

### Example: ASCII – Binary Conversion

- Input: ASCII Hex character
- Output: Binary value

signal input : std\_logic\_vector(7 downto 0);  
 signal output : std\_logic\_vector(3 downto 0);  
 process (input)  
 begin

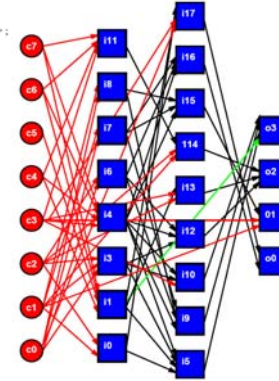
0	NUL	0x00	0	0	0	0	0
1	SOH	0x01	1	A	Q	4	q
2	STX	0x02	2	B	R	6	r
3	ETX	0x03	3	C	S	8	s
4	EOF	0x04	4	D	T	0	t
5	ENQ	0x05	5	E	U	2	u
6	ACK	0x06	6	F	V	4	v
7	BS	0x07	7	G	W	6	w
8	BE	0x08	8	H	X	8	x
9	HT	0x09	9	I	Y	0	y
A	LF	0x0A	10	J	Z	2	z
B	VT	0x0B	11	K	[	4	{
C	FF	0x0C	12	L	]	6	~
D	CR	0x0D	13	M	_	8	^
E	SO	0x0E	14	N	`	0	_
F	SI	0x0F	15	O	~	2	~

end process;

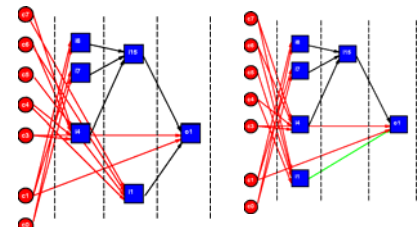
### ASCII – Binary Conversion Circuit

```

INORDER = C<7> C<6> C<5> C<4> C<3> C<2> C<1> C<0>;
OUTORDER = O<3> O<2> O<1> O<0>;
# stage 1 – 8 LUTs [C<3:0> pass through]
i0 = !C<1> + !C<2>;
i1 = C<4> + C<5> + !C<6> + !C<7>;
i3 = C<0> + C<1> + C<2>;
i4 = !C<3> + !C<4> + C<6> + !C<7>;
i6 = !C<0> + C<2>;
i7 = !C<0> + C<1>;
i8 = C<0> + !C<1>;
i11 = !C<7> + C<6> + !C<4> + !C<3>;
# stage 2 – 9 LUTs [i1,C<3>,C<1> pass through]
i5 = i0 * i1 + i3 * i4;
i9 = i6 * i4 + i7 * i4 + i8 * i4;
i10 = C<3> + i3 * i4;
i12 = i3 * i4 + i6 * i4;
i13 = i1 * !C<3> + C<2>;
i14 = C<2> + !C<1> + i11;
i15 = i8 * i4 + i7 * i4;
i16 = i7 * i4 + i6 * i4;
i17 = i1 * !C<3> + C<0> + C<0> + i0 * i1;
# stage 3 – 4 LUTs
O<3> = (i10+i9)*(i5+i9);
O<2> = i12 + i13 + i14;
O<1> = i1 * !C<3> + C<1> + i15;
O<0> = i16 + i17;
    
```

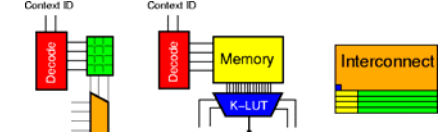


### Implementation Choices



- Both require same amount of execution time
- Implementation #1 more resource efficient

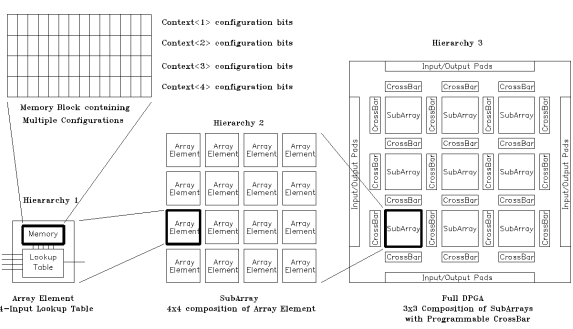
### Previous Study [Deh96B]



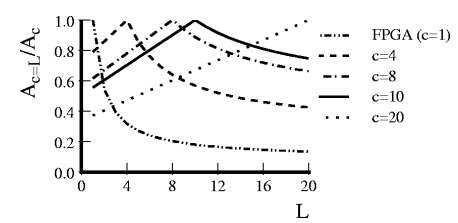
- Interconnect Mux
- Logic Reuse
- $A_{ctxi} \approx 80K\lambda^2$
- dense encoding**
- $A_{base} \approx 800K\lambda^2$
- \* Each context not overly costly compared to base cost of wire, switches, IO circuitry

Question: How does this effect scale?

### DPGA Prototype



### Multicontext Tradeoff Curves



- Assume ideal packing:  $N_{active} = N_{total}/L$
- Reminder:  $c * A_{ctx} = A_{base}$
- Difficult to exactly balance resources/demands
- Needs for contexts may vary across applications
- Robust point where critical path length equals # contexts

## ••• | In Practice

- Scheduling limitations
- Retiming limitations

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.19

## ••• | Scheduling Limitations

- $N_A$  (active)
  - Size of largest stage
- Precedence:
  - Can evaluate a LUT only after predecessors have been evaluated
  - Cannot always, completely equalize stage requirements

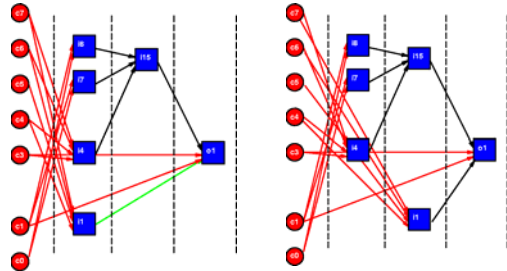
November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.20

## ••• | Scheduling

- Precedence limits packing freedom
- Freedom shows up as slack in network



November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.21

## ••• | Scheduling

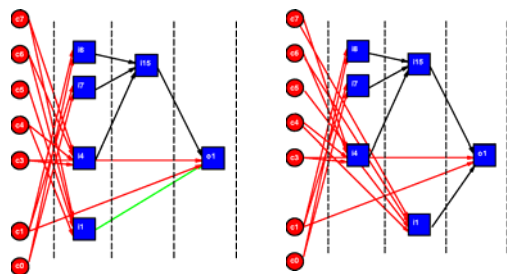
- Computing Slack:
  - ASAP (As Soon As Possible) Schedule
    - propagate depth forward from primary inputs
      - $depth = 1 + \max \text{input depth}$
  - ALAP (As Late As Possible) Schedule
    - propagate distance from outputs back from outputs
      - $level = 1 + \max \text{output consumption level}$
  - Slack
    - $slack = L + 1 - (depth + level)$  [PI depth=0, PO level=0]

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.22

## ••• | Allowable Schedules



Active LUTs ( $N_A$ ) = 3

November 6, 2007

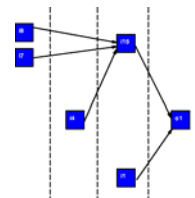
CprE 583 – Reconfigurable Computing

Lect-22.23

## ••• | Sequentialization

- Adding time slots
  - More sequential (more latency)
  - Adds slack
    - Allows better balance

$L=4 \rightarrow N_A=2$  (4 or 3 contexts)



November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.24

## Multicontext Scheduling

- “Retiming” for multicontext
  - **goal:** minimize peak resource requirements
- NP-complete
- List schedule, anneal
  - How do we accommodate intermediate data?
  - Effects?

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.25

## Signal Retiming

- Non-pipelined
  - hold value on LUT Output (wire)
    - from production through consumption
- Wastes wire and switches by occupying
  - For entire critical path delay L
  - Not just for  $1/L$ th of cycle takes to cross wire segment
- How will it show up in multicontext?

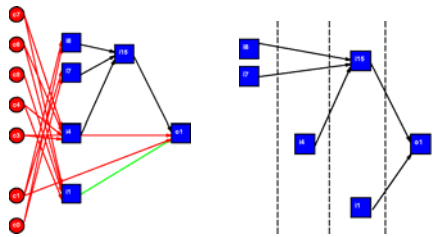
November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.26

## Signal Retiming

- Multicontext equivalent
  - Need LUT to hold value for each intermediate context



November 6, 2007

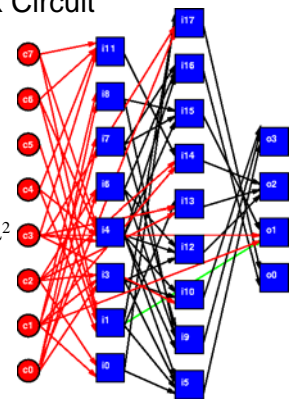
CprE 583 – Reconfigurable Computing

Lect-22.27

## Full ASCII → Hex Circuit

- Logically three levels of dependence
- Single Context:

21 LUTs @  $880K\lambda^2 = 18.5M\lambda^2$



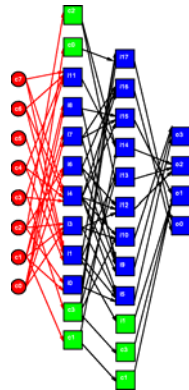
November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.28

## Multicontext Version

- Three contexts:
- 12 LUTs @  $1040K\lambda^2 = 12.5M\lambda^2$
- Pipelining needed for dependent paths



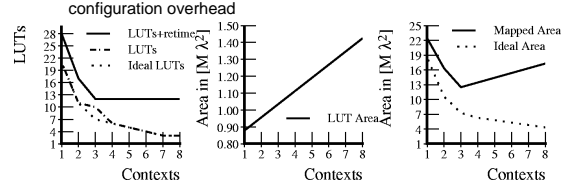
November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.29

## ASCII → Hex Example

- All retiming on wires (active outputs)
  - Saturation based on inputs to largest stage
  - With enough contexts only one LUT needed
  - Increased LUT area due to additional stored configuration information
  - Eventually additional interconnect savings taken up by LUT configuration overhead



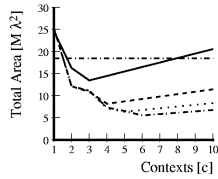
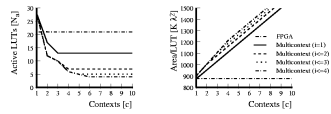
Ideal = Perfect scheduling spread + no retime overhead

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.30

### ASCII→Hex Example (cont.)



@ depth=4, c=6: 5.5M<sup>2</sup> (compare 18.5M<sup>2</sup>)

### General Throughput Mapping

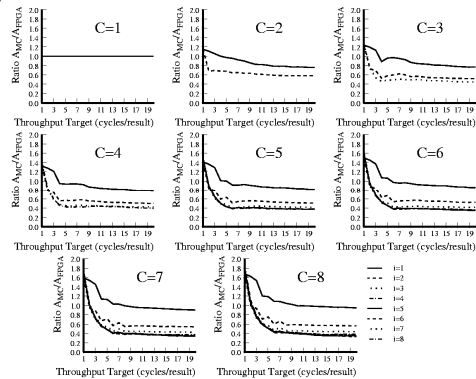
- If only want to achieve limited throughput
- Target produce new result every t cycles
- Spatially pipeline every t stages
  - cycle = t
- Retime to minimize register requirements
- Multicontext evaluation w/in a spatial stage
  - Retime (list schedule) to minimize resource usage
- Map for depth (i) and contexts (c)

### Benchmark Set

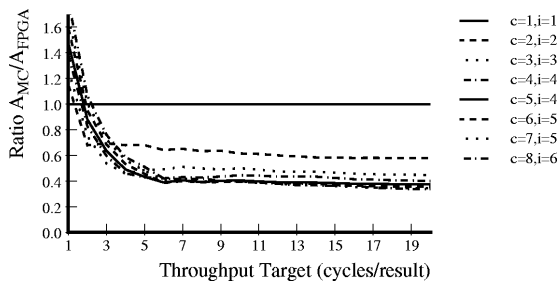
- 23 MCNC circuits
  - Area mapped with SIS and Chortle

Circuit	Mapped LUTs	Path Length	Circuit	Mapped LUTs	Path Length
5xp1	46	10	des	1267	13
9sym	123	7	e64	230	9
9symml	108	8	f51m	45	17
C499	85	10	misex1	20	6
C880	176	21	misex2	38	8
alu2	169	19	rd73	105	10
apex6	248	9	rd84	150	9
apex7	77	7	rot	293	16
b9	46	7	sao2	73	9
clip	121	9	vg2	60	9
cordic	367	13	z4ml	8	7
count	46	16			

### Area v. Throughput



### Area v. Throughput (cont.)



### Reconfiguration for Fault Tolerance

- Embedded systems require high reliability in the presence of transient or permanent faults
- FPGAs contain substantial redundancy
- Possible to dynamically “configure around” problem areas
- Numerous on-line and off-line solutions

## Column Based Reconfiguration

- Huang and McCluskey
- Assume that each FPGA column is equivalent in terms of logic and routing
  - Preserve empty columns for future use
  - Somewhat wasteful
- Precompile and compress differences in bitstreams



Figure 2: Configuration frame architecture in Xilinx Virtex-series FPGAs.

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.37

## Column Based Reconfiguration

- Create multiple copies of the same design with different unused columns
- Only requires different inter-block connections
- Can lead to unreasonable configuration count

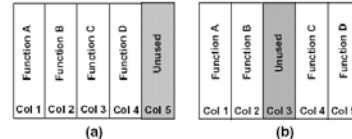


Figure 4: The overlapping precompiled configuration. (a) Base configuration. (b) Alternative configuration when column 3 is intentionally unused.

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.38

## Column Based Reconfiguration

- Determining differences and compressing the results leads to “reasonable” overhead
- Scalability and fault diagnosis are issues

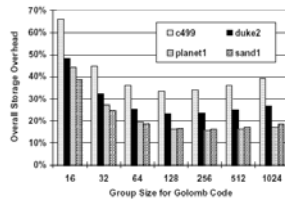


Figure 7: Storage overhead for the overlapping precompiled configuration scheme.

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.39

## Summary

- In many cases cannot profitably reuse logic at device cycle rate
  - Cycles, no data parallelism
  - Low throughput, unstructured
  - Dissimilar data dependent computations
- These cases benefit from having more than one instructions/operations per active element
- Economical retiming becomes important here to achieve active LUT reduction
- For  $c=[4,8]$ ,  $l=[4,6]$  automatically mapped designs are 1/2 to 1/3 single context size

November 6, 2007

CprE 583 – Reconfigurable Computing

Lect-22.40