

# Phased Logic

Automated Clocked-to-Clockless

Robert Reese – Mississippi State Univ.  
(reese@ece.msstate.edu)

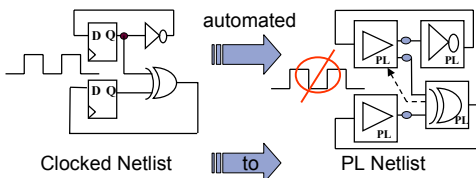
Mitchell A. Thornton – Southern Methodist Univ.  
(mitch@engr.smu.edu)

Cherrice Traver – Union College  
(mitch@engr.smu.edu)

# Outline

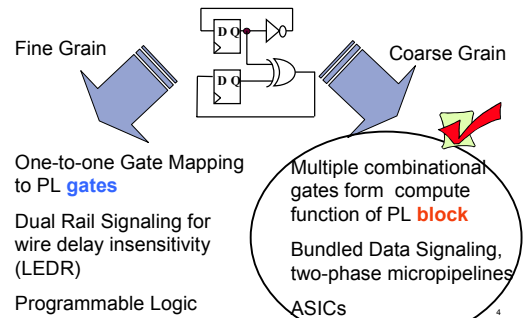
- Methodology Overview ←
- Methodology Details
- Use of Early Evaluation
- Misc Topics
  - CPU register file interface
  - Slack buffers
  - Tutorial Designs
- Fine Grain Methodology

# Phased Logic (PL)



- PL netlist is self-timed, can be faster than clocked netlist via early evaluation, removes global clock.

# Fine Grain vs. Coarse Grain



One-to-one Gate Mapping to PL gates

Dual Rail Signaling for wire delay insensitivity (LEDR)

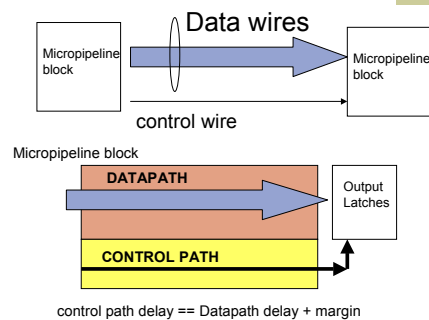
Programmable Logic

Multiple combinational gates form compute function of PL block  
Bundled Data Signaling, two-phase micropipelines  
ASICs

# PL Genealogy

- Micropipelines (Sutherland '88, Turing Award Lecture)
  - Linear, fork/join pipelines, bundled data signaling, no methodology for mapping of complex systems
- LEDR Micropipelines (Dean, Williams, Dill, ARVLSI '91)
  - Level-Encoded Dual-Rail (LEDR) signaling for delay insensitivity
  - Non-general eager evaluation (AND gate)
- Phased Logic (Linder, Harden IEEE TCOMP '96)
  - Generalized, automated mapping of complex clocked systems to LEDR micropipelines
- Fine-grain PL with Early Evaluation
  - Arithmetic structures with early-evaluation (Reese, Thornton, Traver, ICCCD'2001)
  - Generalized early evaluation for improving performance Fine-grain PL (Thornton, Reese, Traver, DATE'2002)
- Coarse-grain PL (Reese, Thornton, Traver, ASYNC'2003)
  - First coarse-grain example, 5-stage CPU, ideal LUT4 technology

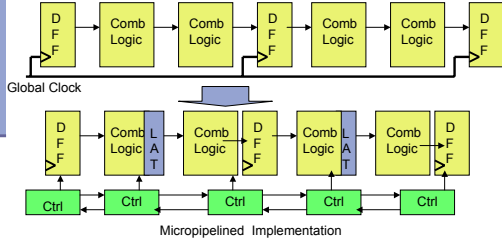
# Micropipelines & Bundled Data



## The Transformation

Based on marked graph theory (Linder/Harden 1996).

Replaces global clock network with distributed control, original logic is **UNTOUCHED**



April 2004

7

## Distinguishing Features

- **Early Evaluation**
  - Block can fire before all inputs arrive
  - Allows PL designs to outperform clocked equivalent (sometimes...)
- **Automated Clocked netlist to self-timed netlist Transformation**
  - Clocked netlist produced by normal synthesis flow, compatible with commercial standard cell libraries, no massive re-invention of CAD tool flow
  - Allow apples-to-apples comparisons of clocked design to asynchronous design

April 2004

8

## Sample Designs

- Double Precision floating point clipping operation, mapped to UMC 0.18 $\mu$ , IBM 0.13  $\mu$ 
  - Several variations
  - Mapped to using coarse-grain and fine-grain flows
- 5-stage pipeline CPU MIPS subset (integer only), mapped to IBM 0.13 $\mu$

April 2004

9

## Floating Point Clip Operation

- Double precision floating-point clipping operation
  - Sign bits used for early evaluation in multi-cycle variations
  - Both sign bits and exponent field calculation used for early evaluation in pipelined version.

```

If ( A < low_bound ) then
    Y = low_bound
else if ( A > high_bound ) then
    Y = high_bound
else Y = A;
    
```

April 2004

10

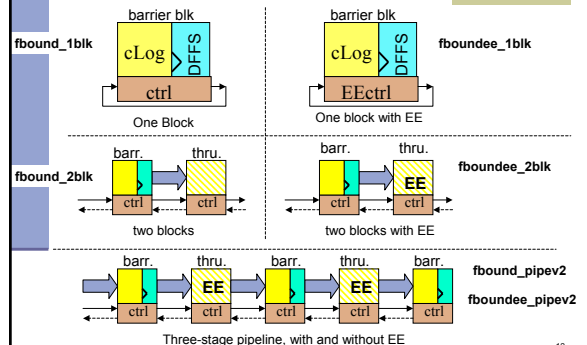
## FP Clip Variations

- Multi-cycle FSM+Datapath as one micropipeline block with and without early evaluation
  - Four states in FSM, one state for loading bounds
  - Result in 2 or 3 clocks
- Multi-cycle FSM+Datapath as two micropipeline blocks with and without early evaluation
- Three-stage pipeline with and without early evaluation
- Mapped to both UMC 0.18 $\mu$ , IBM 0.13  $\mu$
- Test: 1000 vectors, bounds +/- 5.0, vectors random between +/- 15.0.

April 2004

11

## Tutorial Design Variations: FP Clip



April 2004

12

## Results (FP clip, UMC 0.18μ)

Design		finish(ns)	PL/CLK	Min Margin	Min Borrow Mar.
Multi-cycle (1 blk)	clk	12954			
	PL	16013	1.24	22.4%	n/a
Multi-cycle (1 blk)	clk	11630			
	PL (EE)	9334	0.80	19.8%	n/a
Multi-cycle (2 blk)	clk	12461			
	PL	14970	1.20	19.2%	n/a
Multi-cycle (2 blk, EE)	clk	12461			
	PL(EE)	10381	0.83	21.2%	n/a
3-stage pipe	clk	4145			
	PL	5560	1.34	32.4%	32.4%
3-stage pipe	clk	4149			
	PL(EE)	3411	1.82	19.7%	19.7%

April 2004

PL/CLK > 1.0 indicates slowdown

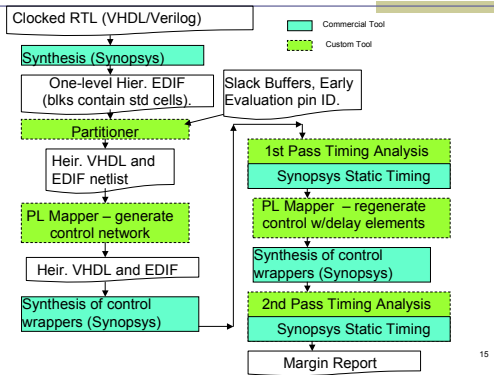
## Results (FP clip, IBM 0.13μ)

Design		finish(ns)	PL/CLK	Min Margin	Min Borrow Mar.
Multi-cycle (1 blk)	clk	5607			
	PL	6597	1.18	11.8%	n/a
Multi-cycle (1 blk)	clk	5288			
	PL (EE)	5042	0.95	15.9%	n/a
Multi-cycle (2 blk)	clk	4909			
	PL	5933	1.21	13.6%	n/a
Multi-cycle (2 blk, EE)	clk	4909			
	PL(EE)	5053	1.03	15.6%	n/a
3-stage pipe	clk	1422			
	PL	2087	1.47	16.1%	11.4%
3-stage pipe	clk	1422			
	PL(EE)	1881	1.32	16.1%	15.5%

April 2004

PL/CLK > 1.0 indicates slowdown

## Phased Logic Methodology Flow



April 2004

15

## Designer Responsibilities in PL

- Components at top level RTL define blocks to be encapsulated
  - No automated partitioning from flattened netlist
  - Blocks can either be mixed DFF/combinational or only combinational
  - Slack Buffers can be inserted automatically by partitioning tool by specifying them in an external initialization file
- Early evaluation blocks must be identified, and an output signal identified as the 'early fire' output

April 2004

16

## PL Tool Flow Features

- Incremental
  - Top-level control script only re-creates modified blocks in a design iteration
- Automated time-borrowing
  - Tool will automatically use time-borrowing between blocks if possible
- Support for blocks without a self-timed wrapper, custom  $\mu$ -pipeline wrappers
  - Designer can specify that a block in top-level netlist not have a wrapper, useful for memories/register files which require custom wrappers
- Target timing margin is user-specified – choose your comfort level.
- Number of technology-dependent files is small, making it easier to support a different cell library.

April 2004

17

## Tutorial Environment Setup/Check

```
[Troia]> source /home/reese/pl_release/pl-setup.bash

[Troia]>  tar xzf /home/reese/pl_release/tut_common.tar.gz

[Troia]>  tar xzf /home/reese/pl_release/tut_umc18.tar.gz

[Troia]>  cd tutorials
[tutorials]> ./make_links.pl umc18
[tutorials]>  rm umc18/fbound_1blk/syn/pl_gate/dpathctrl_top_delays.db
[tutorials]> ./regress_tut.pl cadence umc18 new.rpt fbound_1blk
```

Look at this file to see mapping results.

April 2004

18

## Outline

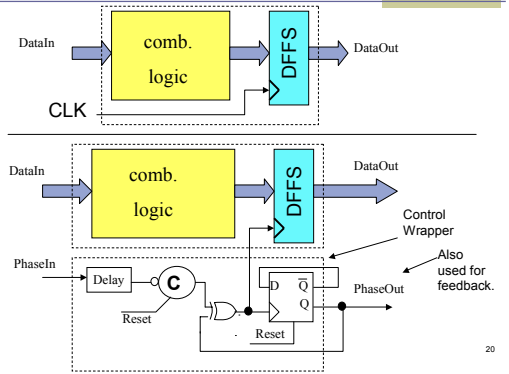
- Methodology Overview
- Methodology Details ←
- Use of Early Evaluation
- Misc Topics
  - CPU register file interface
  - Slack buffers
  - Tutorial Designs
- Fine Grain Methodology

Only highlights from the remaining slides will covered.

April 2004

19

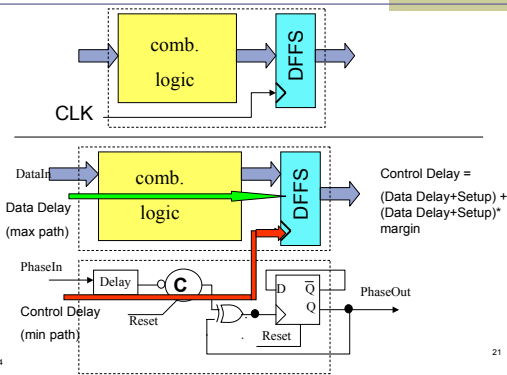
## μpipeline Control Wrapper



April 2004

20

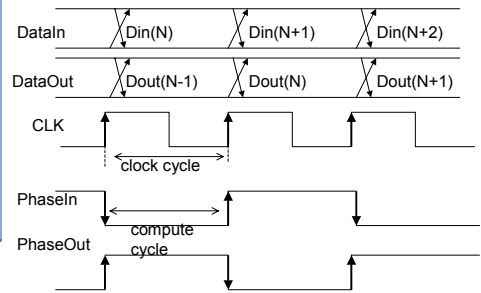
## Delay Matching



April 2004

21

## Two Phase Control

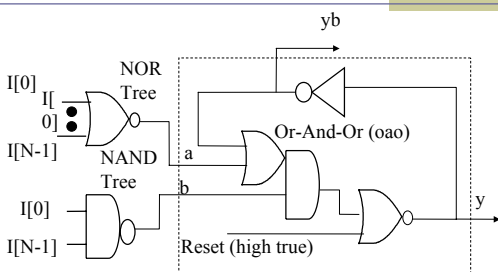


One-to-one correspondence between clock cycle and compute cycle; computation ordering is preserved.

April 2004

22

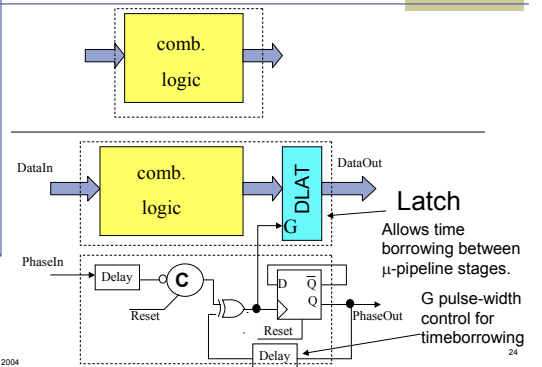
## C-Element Design †



Well suited for large number of inputs, standard cell implementation

† Toy-Yung Wu and Sarma B. K. Vrudhula, "A Design of a Fast and Area Efficient Multi-Input Muller C-element", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol 1, No. 2, June 1993

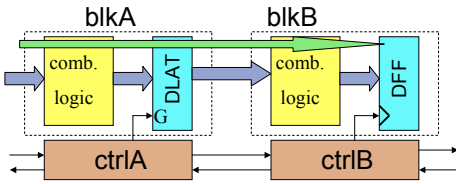
## Combinational-Only Block



April 2004

24

## Time Borrowing



Set delay element in ctrlA, ctrlB such that:  
 $CtrlA + CtrlB \text{ delay} = \text{Total Data Delay} + \text{setup}$

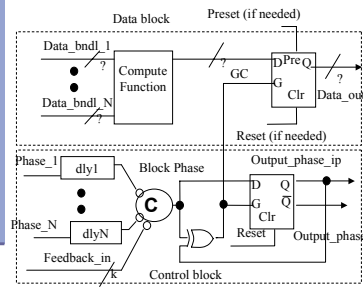
Time borrowing beneficial if  
 $blkB \text{ data delay} < ctrlB \text{ delay}$

This can happen if block B has very thin combinational logic and/or ctrlB logic is slow due to multiple inputs, heavy loading.

April 2004

25

## General Wrapper



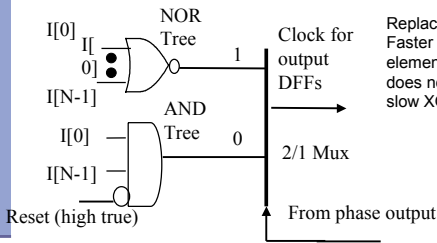
Simply a micropipeline block with two phase control

Phase outputs are also feedback outputs

April 2004

26

## Wrapper Variations: Fast wrapper



Replaces C-element. Faster than old C-element, clock line does not go through slow XOR gate.

The first input arrival after the gate has fired will lower the clock, so cannot use this for time borrowing.

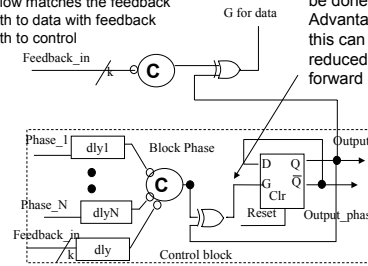
April 2004

27

## Comb. Block Wrapper: Two C-elements

The dly on the feedback inputs below matches the feedback path to data with feedback path to control

Fast version of this can be done as well. Advantage to this is that this can be fast due to reduced loading on forward control path.



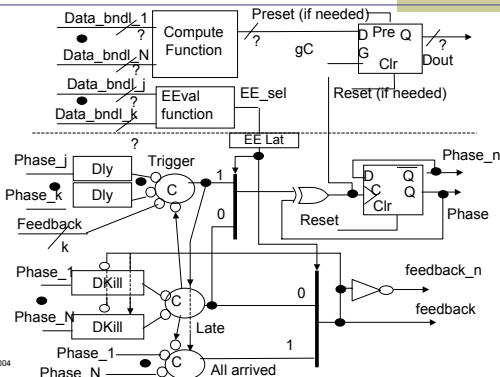
Phase outputs are also feedback outputs

April 2004

28

## Early Evaluation Wrapper

Details on this wrapper later in the talk

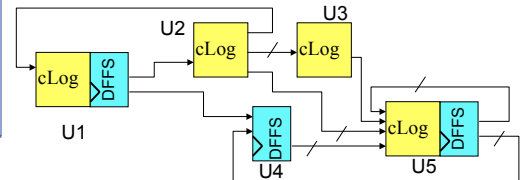


April 2004

29

## Control Network Generation

Given a network of coarse-grain blocks (combinational-only and mixed combinational+DFF), how are control (phase) signals generated?

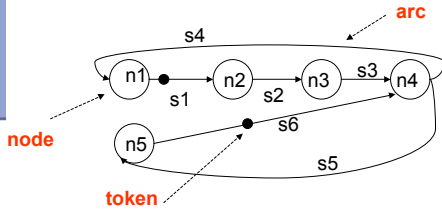


April 2004

30

## Marked Graphs

Control network generation is based on *Marked Graphs*.  
 A **Marked Graph** (MG) is a restricted form of a Petri Net (PN)  
 A MG consists of **nodes**, **arcs**, and an **initial token marking**  
 (the placement of tokens on nodes).



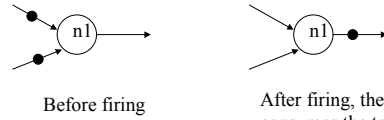
April 2004

31

## Marked Graph Firing Rule

A **token** represents input data.

A node **fires** when there are tokens on all input arcs; firing removes one token (consumes the data) from each input arc and places one token on each output arc.



Before firing

After firing, the node consumes the tokens on its input arcs.

April 2004

32

## Representing Tokens

Each block has an internal state element (**block phase**) whose value is either '0' (EVEN) or '1' (ODD).

If the state of an input (the **input phase**) matches the block phase, then the input has a token (active data).

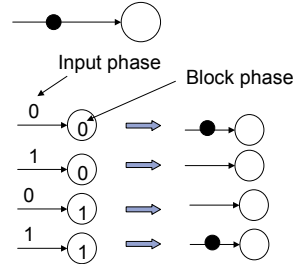
**Firing Rule:** When all input phases match the block phase (all inputs have arrived), then block fires which updates the output and toggles the block phase, 'consuming' the input data.

April 2004

33

## Representing Tokens (cont)

How do you represent a token on an arc?



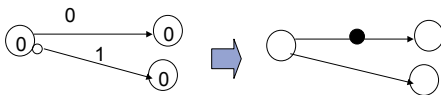
April 2004

34

## Initial Token Marking

To create a live and safe marked graph, must place initial tokens on some arc.

Initial token marking is just a wiring choice !!!



April 2004

35

## Liveness and Safety

A marked graph is **live** if there is always a node somewhere in the graph ready to fire.

A marked graph is **safe** if the token count of an arc never exceeds 1 (i.e. new output data is NOT produced until old output data has been consumed).

A dead system does not circulate data.

A node in an unsafe system can produce new data before its output data has been consumed by its successor nodes, resulting in loss of data.

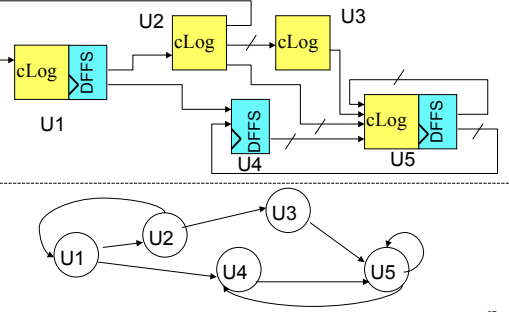
**The control network is mapped to a live and safe marked graph!!!!**

April 2004

36

## Example Control Mapping

Each block represents a node in a marked graph.



April 2004

37

## Creating a Live and Safe MG

For liveness and safety, each fanout must be part of a directed circuit with token count = 1.

Extra signals called **feedbacks** (a.k.a acknowledgements) are added to create these loops (one for each fanout in coarse-grain mapper).

**Token Marking** adds initial tokens to arcs to create a loop with token count = 1.

A mixed DFF+combinational block is called a **Barrier** block.

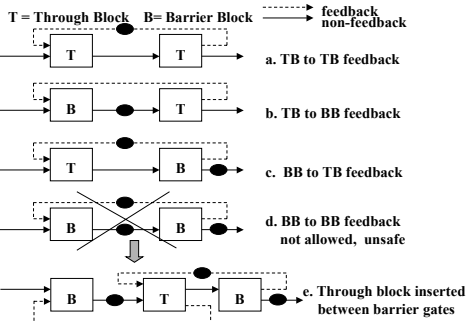
A combinational block is called a **Through** block.

Initial Token marking rules are based on Barrier/Through block connections.

April 2004

38

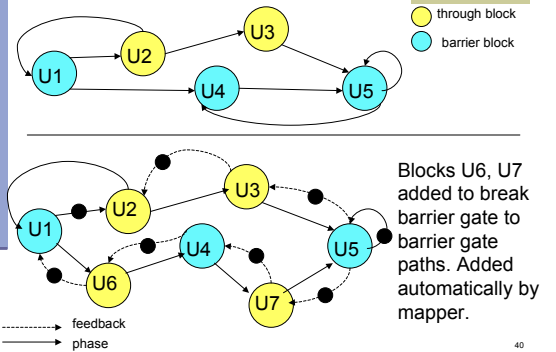
## Initial Token Marking Rules



April 2004

39

## Example Control Generation

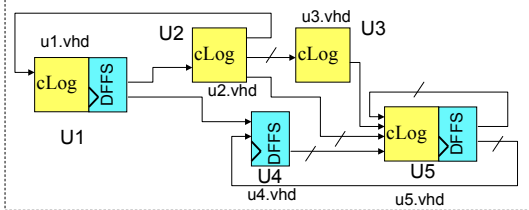


April 2004

40

## Defining the top level blocks

top.vhd

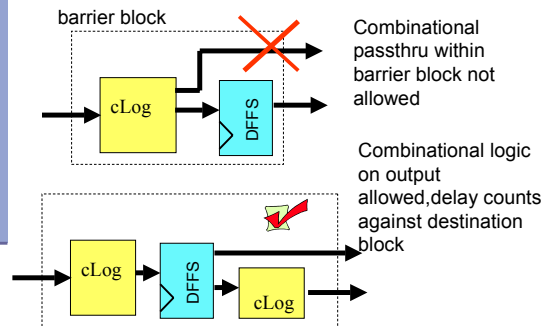


Blocks at top level of netlist determine the blocks that will have wrappers. A block can have internal hierarchy, but it will be flattened during the mapping process.

April 2004

41

## Block restrictions

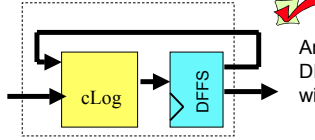


April 2004

42

## Block Restrictions (cont)

barrier block



Any DFF to DFF or DFF to CLogic path within block allowed

This implies that any clocked system without combinational feed-thrus can be encapsulated as a SINGLE barrier block.

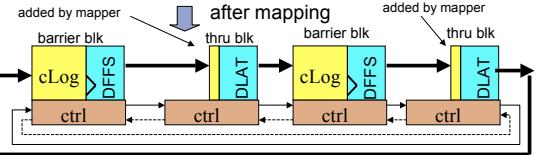
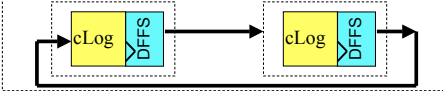
Obviously, this is not satisfactory as the load on the output latch signal has not been reduced at all! Use partitioning to reduce loading on the latch signal.

April 2004

43

## Partitioning affects performance

top.vhd



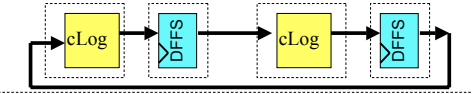
Feedback latency from barrier blk to thru-block to thru-block output adds to cycle time, causing performance penalty

April 2004

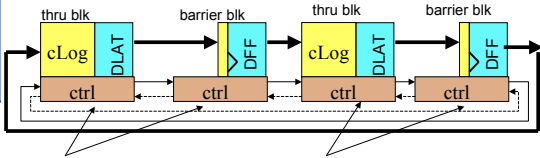
44

## A better partitioning

top.vhd



after mapping

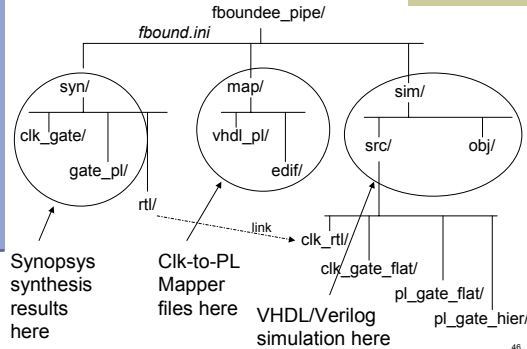


Mapper can now perform time borrowing between these blocks.

April 2004

45

## Clk-to-PL Walkthru: Directory Structure



Synopsys synthesis results here

Clk-to-PL Mapper files here

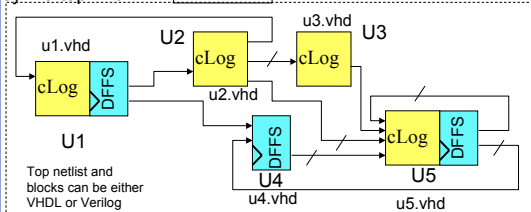
VHDL/Verilog simulation here

April 2004

46

## Clk-to-PL Walkthru: Starting point

syn/rtl/top.vhd → Synopsys → syn/clk\_gate/top\_hier.edif



Top netlist and blocks can be either VHDL or Verilog

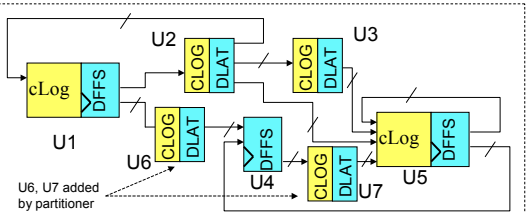
Each block synthesized by Synopsys to a netlist of standard cells. Synopsys then used to produce a hierarchical EDIF netlist which is the input the PL tool flow.

April 2004

47

## Clk-to-PL Walkthru: Partitioner

syn/clk\_gate/top\_hier.edif  
top.ini (mapper options) → Partitioner → map/vhdl\_pl/top.vhd



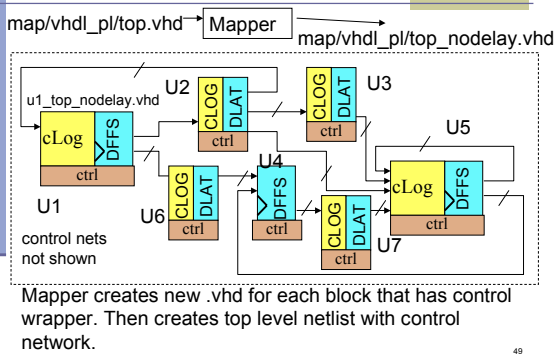
U6, U7 added by partitioner

Partitioner adds buffers between barrier blocks, adds latches to all thru blocks, also prepares blocks for latch/clock inputs to be driven by balanced buffer network.

April 2004

48

## Clk-to-PL Walkthru: Mapper, 1<sup>st</sup> pass



April 2004

49

## Clk-to-PL Walkthru: Wrapper Synthesis

Synthesize each block+control wrapper (only wrapper logic is synthesized, block logic is untouched)

map/vhdl\_pl/u1\_top\_nodelay.vhd

Synopsys

syn/gate\_pl/u1\_top\_nodelay.db

Gate level netlist of block+control wrapper

April 2004

50

## Clk-to-PL Walkthru: Netlist, 1<sup>st</sup> pass

syn/gate\_pl/\*\_nodelay.db ← All 1<sup>st</sup> pass gate-level block files

Synopsys

syn/gate\_pl/top\_nodelay\_flat.db  
syn/gate\_pl/top\_nodelay\_flat.v  
syn/gate\_pl/top\_nodelay\_flat.sdf

Flattened gate-level .db and verilog netlists of PL design with SDF timing

April 2004

51

## Clk-to-PL Walkthru: 1<sup>st</sup> Pass Timing

syn/gate\_pl/top\_nodelay\_flat.db

plcg\_timing.pl  
Synopsys Static Timing

syn/gate\_pl/u1\_top\_nodelay.tim  
syn/gate\_pl/u2\_top\_nodelay.tim  
syn/gate\_pl/u3\_top\_nodelay.tim  
etc...

Timing information extracted for each block.

.tim file contains timing information needed for delay block calculation.

All delay block and margin calculation derived from .tim files.

April 2004

52

## Clk-to-PL Walkthru: .tim File

```
cell bpc1_control bpc1 bpc
flag wrapper wrap
cport none bpcpipe1_control t
cport_delay data 0.43
cport_delay ctrl 0.45
end_cport
cport none ifetch1_control t
cport_delay data 0.96
cport_delay ctrl 0.46
end_cport
cport fb pc1_control_fb f
cport_delay ctrl 0.36
cport_delay data 0.43
end_cport
cport out bpc1_control
end_cport
cport outfb bpc1_control_fb
end_cport
end_cell
```

wrapper type

input port, non-early eval

src is ifetch1 block

datapath delay

control path delay

control path < data path  
so delay blocks required

April 2004

53

## Clk-to-PL Walkthru: Delay Calculations

syn/gate\_pl/\*\_nodelay.tim ← (all 1<sup>st</sup> pass timing files)

plcg\_timing.pl

input\_dly entries in top\_delays.ini file

top\_delays.ini

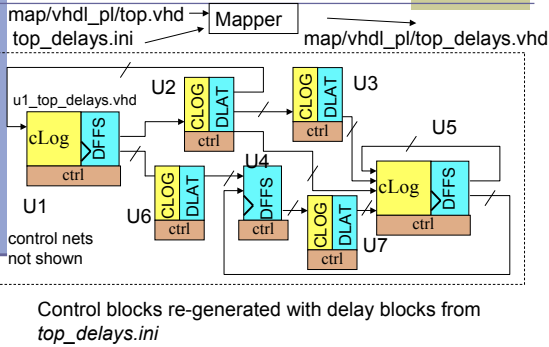
```
input_delay bpcpipe1 bpc1 4 ; ###autogen
input_delay ifetch1 bpc1 16 ; ###autogen
```

Number of delay blocks for this input.

April 2004

54

## Clk-to-PL Walkthru: Mapper, 2<sup>nd</sup> pass



April 2004

55

## Clk-to-PL Walkthru: Wrapper Synthesis, 2<sup>nd</sup> pass

Synthesize each block+control wrapper (only wrapper logic is synthesized, block logic is untouched)

map/vhdl\_pl/u1\_top\_delays.vhd

Synopsys

syn/gate\_pl/u1\_top\_delays.db

Gate level netlist of block+control wrapper with delay blocks

April 2004

56

## Clk-to-PL Walkthru: Netlist, 2<sup>nd</sup> pass

syn/gate\_pl/\*\_delays.db

All 2nd pass gate-level block files

Synopsys

syn/gate\_pl/top\_delays\_flat.db  
 syn/gate\_pl/top\_delays\_flat.v  
 syn/gate\_pl/top\_delays\_flat.sdf  
 syn/gate\_pl/top\_delays\_hier.v  
 syn/gate\_pl/top\_delays\_hier.sdf

Final gate-level .db and verilog netlists of PL design with SDF timing. Flat netlist for timing information and standard cell layout, hierarchical netlist for simulation/debugging.

April 2004

57

## Clk-to-PL Walkthru: 2<sup>nd</sup> Pass Timing

syn/gate\_pl/top\_delays\_flat.db

Timing information extracted for each block.

plcg\_timing.pl  
 Synopsys Static Timing

syn/gate\_pl/u1\_top\_delays.tim  
 syn/gate\_pl/u2\_top\_delays.tim  
 syn/gate\_pl/u3\_top\_delays.tim  
 etc...

Blocks re-characterized so that margins can be calculated.

April 2004

58

## Clk-to-PL Walkthru: .tim File

```
cell bpc1_control bpc1 bpc
flag wrapper wrap
cport none bpcpipe1_control t
cport_delay data 0.43
cport_delay ctrl 0.70
end_cport
cport none ifetch1_control t
cport_delay data 0.96
cport_delay ctrl 1.26
end_cport
cport fb pc1_control_fb f
cport_delay ctrl 0.56
cport_delay data 0.42
end_cport
cport out bpc1_control
end_cport
cport outfb bpc1_control_fb
end_cport
end_cell
```

wrapper type

input port, non-early eval

src is ifetch1 block

datapath delay

control path delay

control path delay is now greater than datapath delay

April 2004

59

## Clk-to-PL Walkthru: Margin Report

syn/gate\_pl/\*\_delays.tim (all 2<sup>nd</sup> pass timing files)

plcg\_timing.pl

map/top\_delays\_margin.rpt contains margin report

top\_delays\_margin.rpt

Margin	#dlyblks	Source	Destination
34.9%	4	bpcpipe1 (barr)	bpc1 (thru)
17.7%	14	ifetch1 (barr)	bpc1 (thru)

margin

delay blocks

April 2004

60

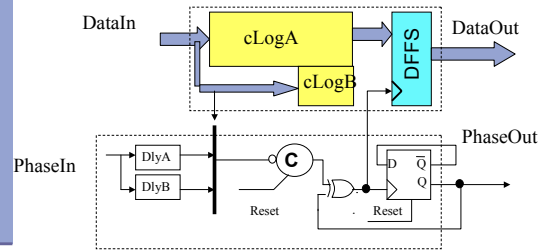
## Outline

- Methodology Overview
- Methodology Details
- Use of Early Evaluation ←
- Misc Topics
  - CPU register file interface
  - Slack buffers
  - Tutorial Designs
- Fine Grain Methodology

April 2004

61

## Bypass Operation for $\mu$ -pipeline block



Bypass operation for a  $\mu$ -pipeline block chooses different delay based upon operation. Takes advantage of short paths, cycle computation is now data dependent.

April 2004

62

## Early Evaluation

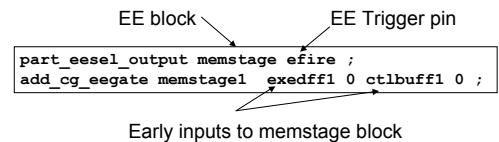
- Early evaluation is a generalized form of bypass operation
- Inputs divided into two classes
  - Early arriving
  - Late arriving
- Two Types of fires
  - Early Fire: Block fires when all early inputs have arrived *and* early evaluation trigger function is true (EFIRE = 1)
  - Late Fire: Block fires when all inputs have arrived

April 2004

63

## Early Eval: Designer Responsibilities

- Identify blocks that will have early evaluation
- Identify early inputs
- Identify output pin that will function as the early evaluation trigger (EFIRE), modify RTL to produce EFIRE signal
- identify pins from source block or path-gates within compute function to assist static path timing analysis (optional)
- Information contained in *design.ini* file.

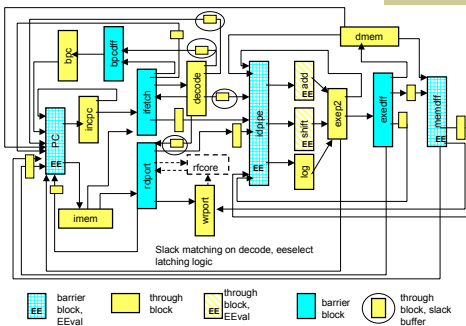


Early inputs to memstage block

April 2004

64

## Early Evaluation: MIPS-subset CPU



April 2004

65

## Early Evaluation: MIPS-subset CPU

- PC stage
  - PC+1 identified as 'early' input; Branch/Jump result as late input due to extra latency due to forwarding logic
- Memstage
  - Input from data memory classified as 'late', all other inputs early. Early evaluated whenever results from data memory not required
- ALU
  - Split into three blocks: Add/Sub, Barrel Shift, other. Add/Sub, Barrel Shift bypassed when not needed
- Idecode
  - Forwarded results from ALU and data memory were late inputs

April 2004

66

## Early Evaluation: RTL Example

```
-- multiplex next pc
efire <= not(in_ctrl_if.branch or in_ctrl_if.jump);

-- don't know what baddr_sel will be on efire
-- so make sure it is a don't care via the
-- use if.branch and if.jump
nextpc <= branch_addr when
  ((baddr_sel = '1') and
  ((in_ctrl_if.branch='1') or (in_ctrl_if.jump='1')))
  else newpc;
```

Provided by instruction decode as early inputs to PC block

branch\_addr provided by branch pc block as late input

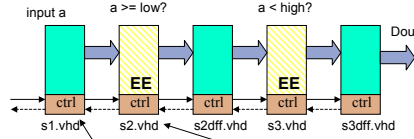
newpc provided by increment pc block as early input

April 2004

67

## Early Eval : DP FP Clip Operation

### 3-stage pipeline



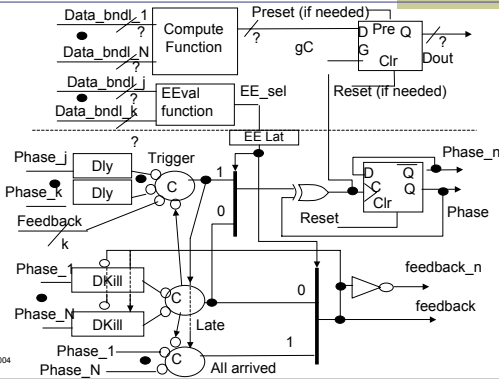
Stage 1 pre-computed  $a \geq \text{low\_bound}$  based on sign and exponent fields. Mantissa comparison in stage 2 bypassed if not needed

Stage 2 pre-computed  $a < \text{high\_bound}$  based on sign and exponent fields. Mantissa comparison in stage 3 bypassed if not needed or if  $a < \text{low}$  (high bound comparison not needed).

April 2004

68

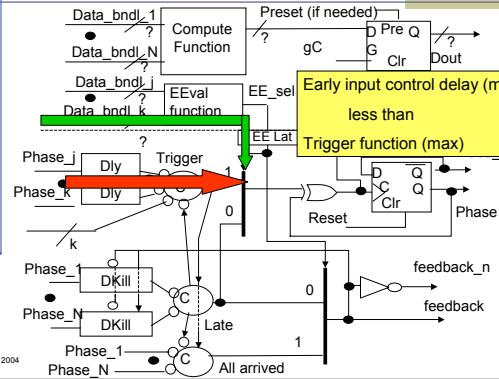
## Early Evaluation Wrapper



April 2004

69

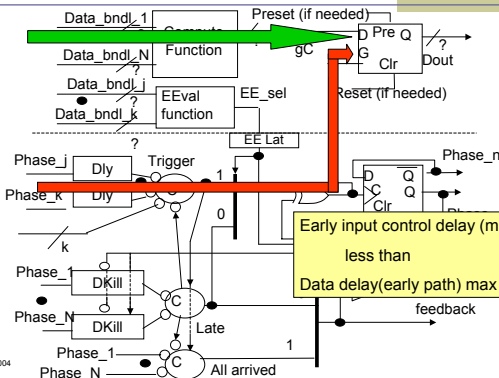
## EE Wrapper: Delay Constraints



April 2004

70

## EE Wrapper: Delay Constraints (cont.)



April 2004

71

## Early Eval: Path Identification

Control input from a block can be both an early and late input.

In this case, give directions in .ini file to assist static path analysis.

Starting pins from block S1 to trace early-eval data path from through block S2

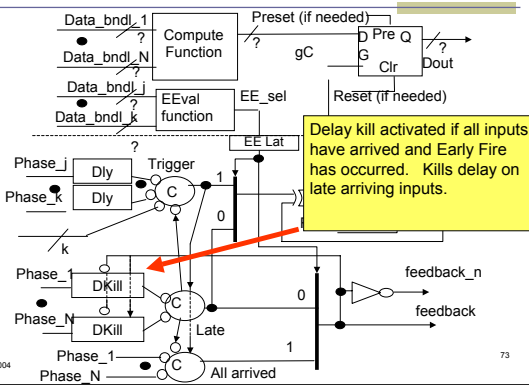
```
cg_eegate_dpath_startpin s1 s2 a_lt_b sign_gt;
```

Can also specify list of gates that the target path passes through in the destination block.

April 2004

72

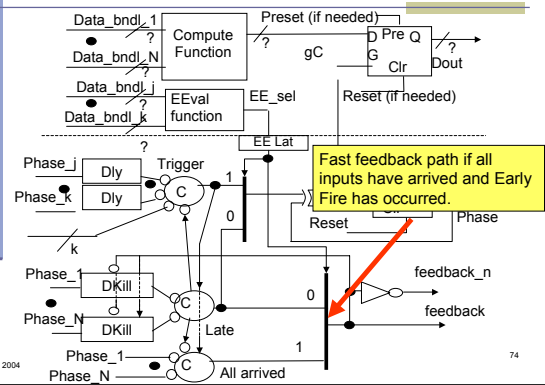
## EE wrapper: Delay Kill



April 2004

73

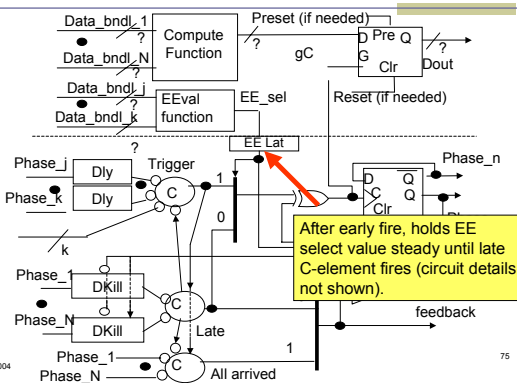
## EE wrapper: Fast Feedback



April 2004

74

## EE wrapper: EE Select (EFIRE) Latch



April 2004

75

## Outline

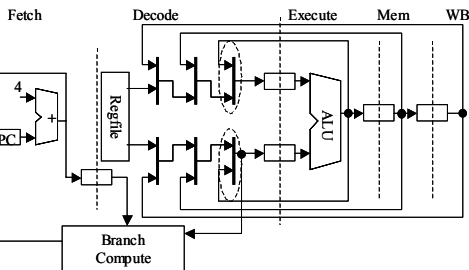
- Methodology Overview
- Methodology Details
- Use of Early Evaluation
- Misc Topics
  - CPU register file interface
  - Slack buffers
  - Tutorial Designs
- Fine Grain Methodology

April 2004

76

## MIP-subset Pipelined CPU

- MIPS integer subset (no multiply)
- Early evaluation used in forwarding paths



April 2004

77

## Code optimization for early evaluation

```

addi r4,r4,1
slli r2,r2,8
bne r2,r0,L10
    
```

Forwarding needed, execute operand mux cannot early eval.

Instruction Reorder

```

slli r2, r2, 8
addi r4,r4,1
bne r2, r0, L10
    
```

Functionally equivalent, but forwarding no longer needed.

Execute operand mux can early evaluate.

'gcc -O' use for compilation of C benchmarks

April 2004

78

## Results (CPU, IBM 0.13μ)

Benchmark		PL/CLK
Fibonacci	n/a	1.12
Bubble Sort		1.16
	Code opt	1.15
CRC		1.12
	Code opt	1.08
Sieve		1.18
	Code opt	1.15
Matrix Tpose		1.19
	Code opt	1.16

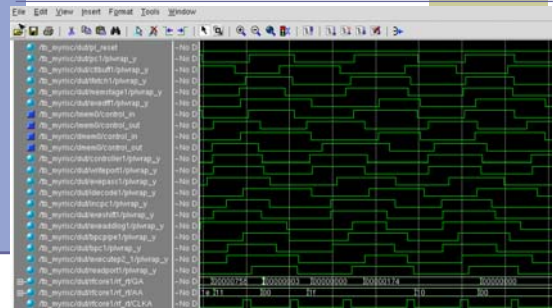
Minimum margin, Minimum Borrow margin: > 10%

April 2004

PL/CLK > 1.0 indicates slowdown

79

## Processor Execution Snapshot



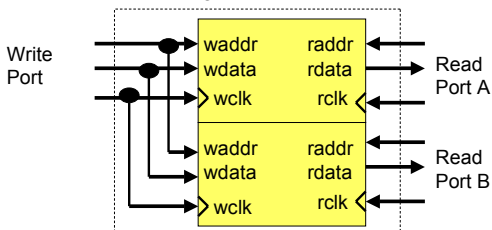
Control signals are shown, note that edge transitions are spread out.

April 2004

80

## MIPS-subset CPU

2-readport, 1-write port 32x32 register file built from dualport RAM generator

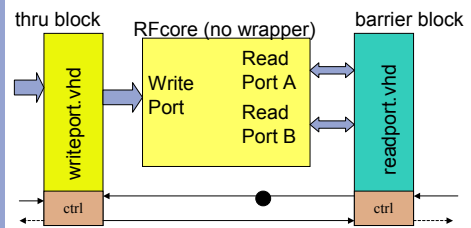


Ports edge-triggered, time-separation constraint between read and write operations

April 2004

81

## Register File Interface

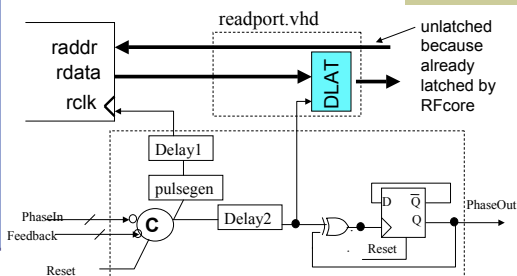


Both readport and writeport used the same custom wrapper (*trigwrap*). Entries in *.ini* file used to specify custom wrapper, non-standard net connections between wrapper and compute block, and that the RFCORE was to have no wrapper.

April 2004

82

## trigwrap for Regfile Interface



Delay1 matched to **setup time** requirement, Delay2 to **access time** requirement (not entire time as time borrowing was used in destination block). Write port done similarly.

April 2004

83

## .ini File Entries for RF port interface

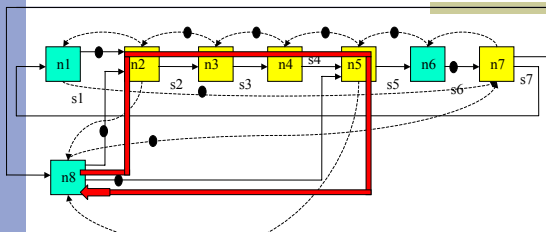
```
part_add_cell_attribute rfcore no_control 1 ; RFCORE has no wrapper
part_add_cell_attribute readport allow_feedthru 1 ; allow unlatched outputs on readport
part_add_pin_attribute readport clka no_outlatch 1 ;
part_add_pin_attribute readport rfcore_rs_4 no_outlatch 1 ;
add_proplist readport1 wrapper trigwrap 0 ; wrapper types for readport, writeport
add_proplist writeport1 wrapper trigwrap 0 ;
add_proplist readport1 local_net g_trig clka_in ; non-standard wrapper connections for readport, writeport
add_proplist readport1 generic trig_dly 4 ; wrapper delays for readport
add_proplist readport1 generic pwidth 8 ;
add_proplist readport1 generic out_dly 12 ;
add_proplist readport1 generic gwidth 12 ;
```

Support exists for custom wrappers to provide **user flexibility**.

April 2004

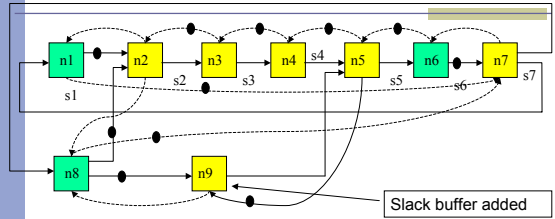
84

## Average Cycle Time (cont.)



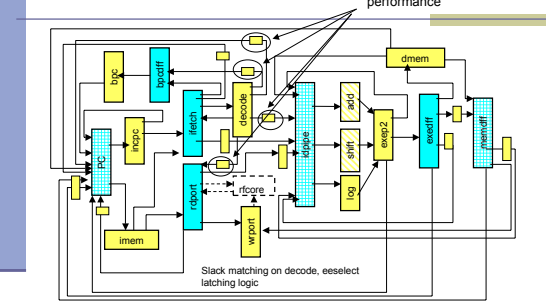
Directed circuit with highest average cycle time is  $n8 > n2 > n3 > n4 > n8$ ,  
 Total delay is 50, token count = 1.  
 Average cycle time =  $50/1 = 50$

## Improve Cycle Time: Slack Buffer



Directed circuit with highest average cycle time is now  $n1 > n2 > n3 > n4 > n5 > n6 > n7$   
 Total delay is 70, token count = 2.  
 Average cycle time =  $70/2 = 35$   
**Performance improved by adding slack buffer!**

## Slack Buffers: CPU



`add_buffer ctlbuff1 pcl 2 1;`  
`add_buffer ctlbuff1 readport1 2 1;`  
`add_buffer ctlbuff1 idecode1 2 1;`

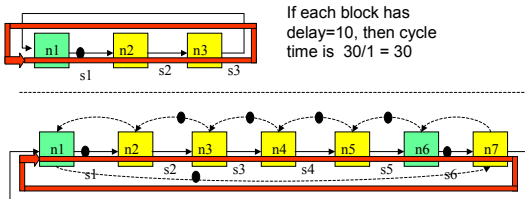
Buffers specified in ini file, are added by mapper to netlist, do not have to specify in clocked RTL (all buffers not shown)

## Porting to a new cell library

- A VHDL package called *tech\_gates* isolates wrappers from a particular library
  - Wrappers have 'fixed' gates (don't touch gates) for better synthesis results and for known timing start/end points
- Only a few core macros (e.g. C-element) are technology dependent
- Two files called *tech.ini* and *gatelib.def* specifies technology dependent information to the mapper
  - gate/pin information
  - delay element base time

## Average Cycle Time

Performance in a marked graph is bounded by the longest directed circuit cycle time divided by the number of tokens on the directed circuit.



Directed circuit with highest average cycle time is  $n1 > n2 > n3 > n4 > n5 > n6 > n7 > n1$ , total delay is 70. This circuit has 2 tokens, so average cycle time is  $70/2 = 35$ .

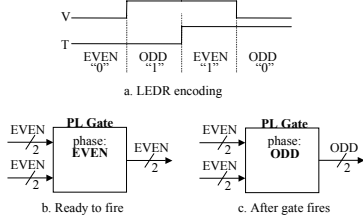
## Outline

- Methodology Overview
- Methodology Details
- Use of Early Evaluation
- Misc Topics
  - CPU register file interface
  - Slack buffers
  - Tutorial Designs
- Fine Grain Methodology ←

## Fine-Grain Methodology

Level-Encoded Dual-Rail (LEDR) Signaling used between PL gates for delay insensitivity.

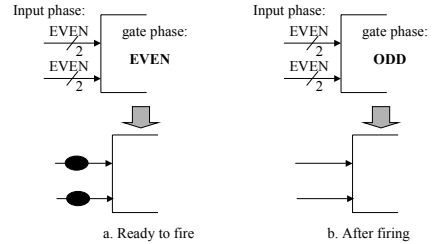
Each PLgate implements 4-input lookup table (LUT4) function.



April 2004

91

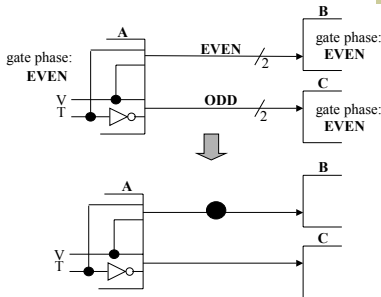
## Token Abstraction



April 2004

92

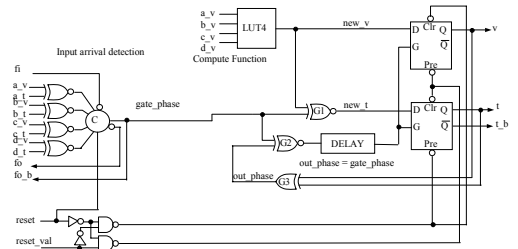
## Initial Token Marking



April 2004

93

## PL4Gate (no early evaluation)



April 2004

94

## Example Results

Design	Variation	cells	PL/clk (area)	Clk cyc	finish time	PL/clk (perf)	FB fanout	FB Max Len
fbound_tblk(clk)	--	2022	--	49.4	141003			
fbound_tblk(PL)	noee_plen1	2257	1.12	n/a	179781	1.28	4700	1
	noee_plenmax	2257	1.12	n/a	186624	1.32	2113	42
	noee_fbopt	2257	1.12	n/a	182062	1.29	3427	29
	ee_i06_c50	3094	1.53	n/a	85902	0.61	4700	1
	ee_max	3220	1.59	n/a	81026	0.57	4700	1
fbound_pipe(clk)	--	1512	--	41.2	41530			
fbound_pipe(PL)	noee_plen1	1933	1.28	n/a	28934	0.7	3465	1
	noee_plenmax	1933	1.28	n/a	49013	1.16	1587	32
	noee_fbopt	1933	1.28	n/a	28643	0.69	2039	19
	ee_i06_c50	2413	1.6	n/a	25511	0.61	3465	1
	ee_max	2530	1.67	n/a	24044	0.58	3465	1

April 2004

95

## Fine-grain Methodology Features

- Operates from flattened EDIF netlist of clocked RTL
- Early evaluation is done automatically by separate tool that walks the netlist, and inserts early evaluation gates
  - Command line options can be used to control amount of early evaluation
- One tool (*plmap*) is used to perform mapping
  - No static timing analysis needed since signaling is delay insensitive

April 2004

96

## Testing the Fine-grain examples

```
[Troia]> source /home/reese/pl_release/pl-setup.bash
```

```
[Troia]> gtar xzf /home/reese/pl_release/tutfg.tar.gz
```

```
[Troia]> cd tutorials_fg
```

```
[tutorials_fg]> ./make_links.pl lut4
```

```
[tutorials_fg]> ./regress_tut.pl cadence lut4 new.rpt fbound_1blk noee_plen1
```

Look at this file to see mapping results. Execute the script without arguments to see the fine-grain example choices.