Uncle – An RTL Approach To Asynchronous Design

Robert B. Reese (Mississippi State University)

Scott C. Smith (University of Arkansas)

Mitchell A. Thornton (Southern Methodist University)

Outline

- Motivation
- NULL Convention Logic (NCL) background
- NCL Systems
- Uncle Synthesis Flow Details
- Design Examples and Comparisons
- Summary and Future Work

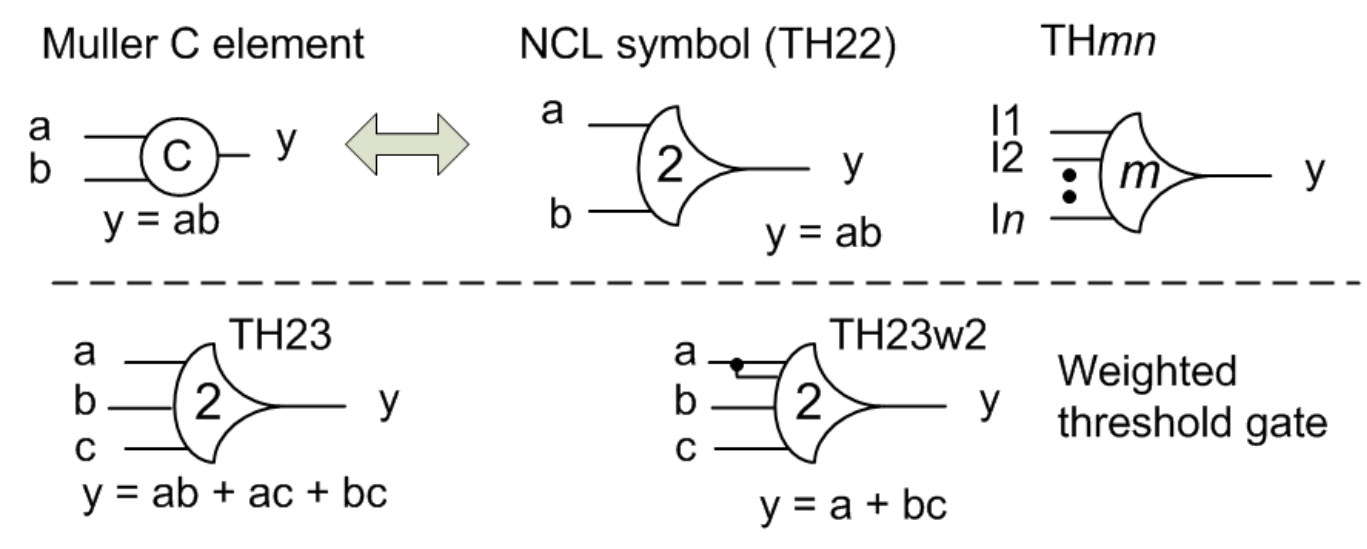
Motivation

- Would like a readily-available asynchronous design flow that
  - Uses a standard RTL (i.e., Verilog/VHDL) so can take advantage of commercial tools for these languages.
  - Should generate a complete system (sequential/combinational logic, datapath+control), have timing analysis, and performance/area optimizations.

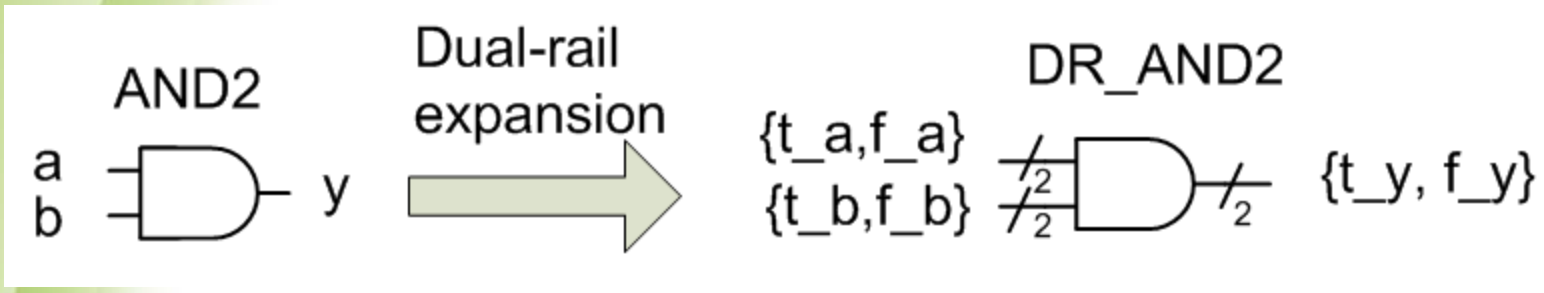
This sounds familiar….

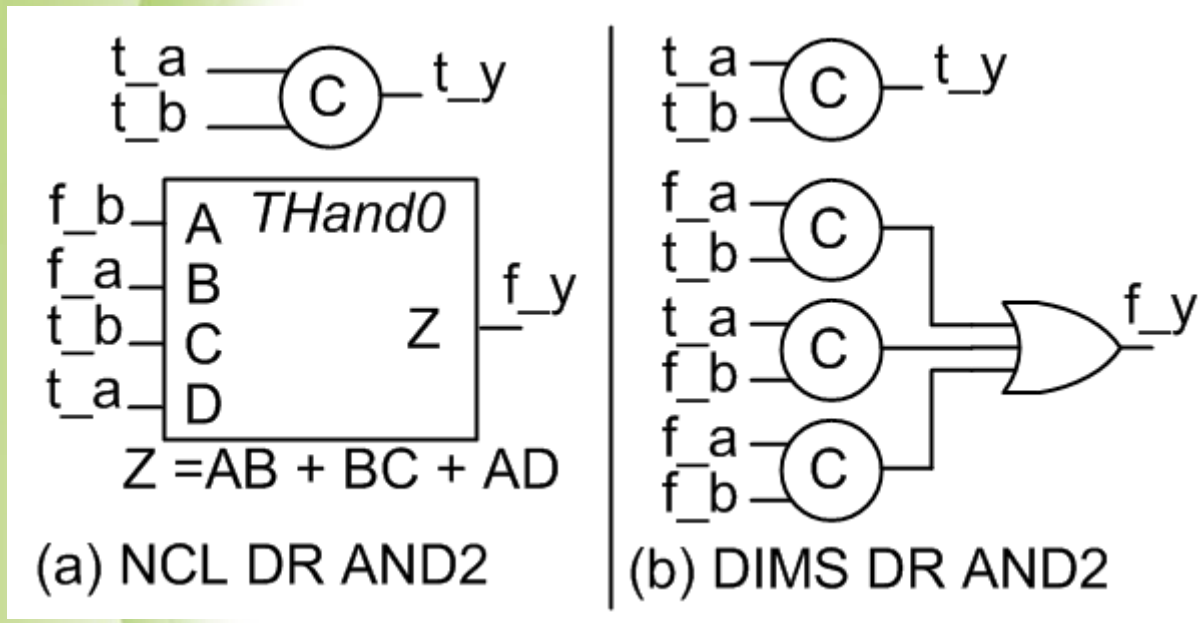
- Theseus Logic flow for *NULL Convention Logic* (circa late 90's-mid 2000s) (Ligthart, Fant, Smith, Taubin, Kondratyev., Async 2000)
  - Used VHDL, Synopsys as front-end.
  - Combinational logic/sequential logic in separate files, ack networks generated manually.
  - Timing tool called *CyclePath* used to measure loop performance, orphan detection.
  - Theseus Logic is now Camgian Microsystems (Maitland/Florida, Starkville/Mississippi).
  - Original flow is unavailable for comparison purposes.
- Reese et.al began work on new flow in December 2010 with goal of synergistic activities with Camgian regarding NCL design (new flow was not solicited by Camgian).

NULL Convention Logic Background

- Four-phase, dual-rail logic family based on threshold logic
  - Can be used to build delay-insensitive systems
  - 27 fundamental gates (all combinations of 2, 3, 4 inputs).
  - CMOS static and semi-static implementations
- THmn threshold gate (at least m inputs of n total inputs asserted before output is asserted).
  - 

Dual-rail Combinational Logic in NCL





31 transistors

56 transistors

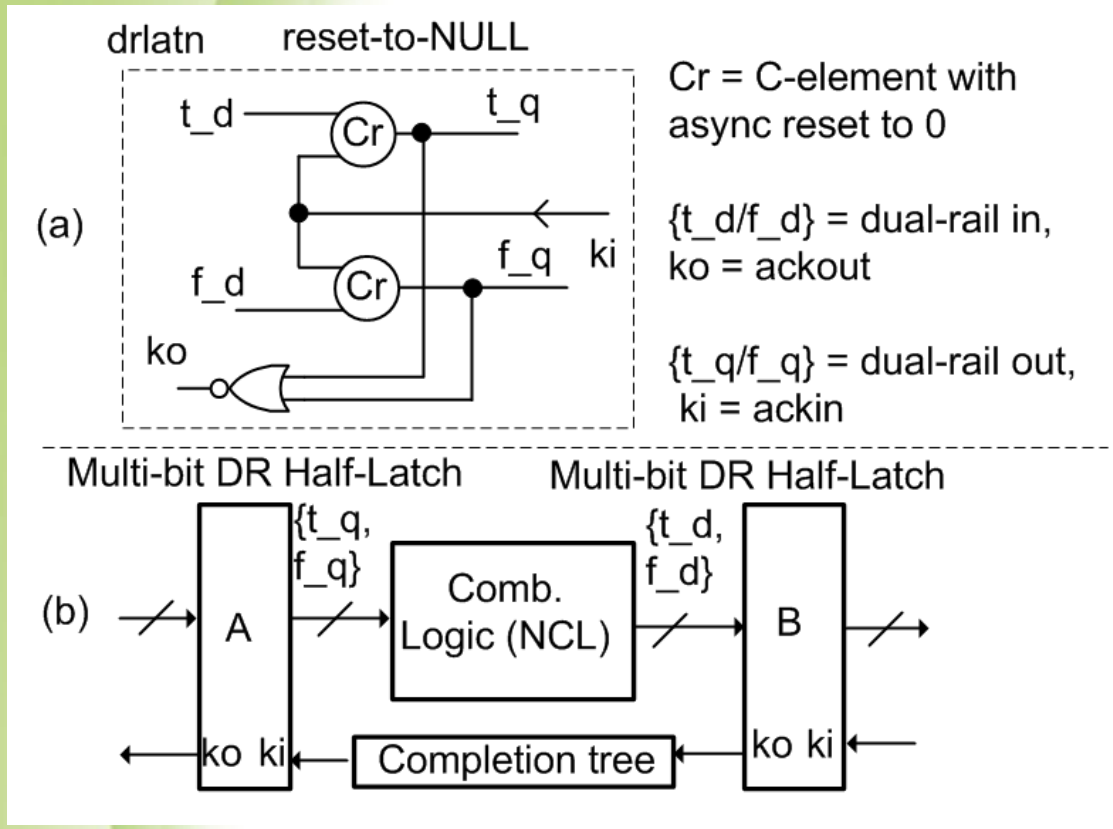
Basic approach for combinational logic is to represent as netlist of AND2, OR2, XOR2, NOT and dual-rail expand the netlist; logic is *input-complete*.

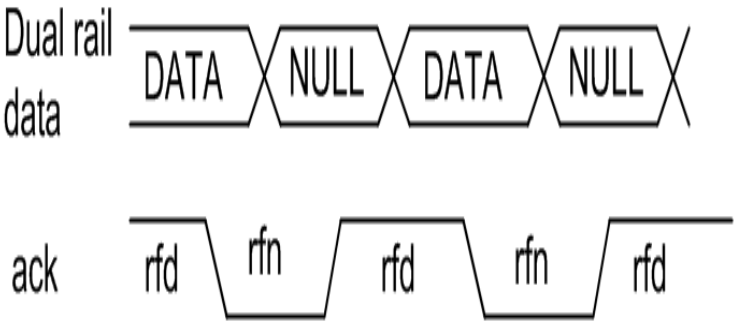
Some complex gates such as MUX2 and FULL ADDER have optimized NCL implementations.

NCL dual-rail more efficient than DIMS

Linear Pipeline

Half-latch, Reset-to-NULL

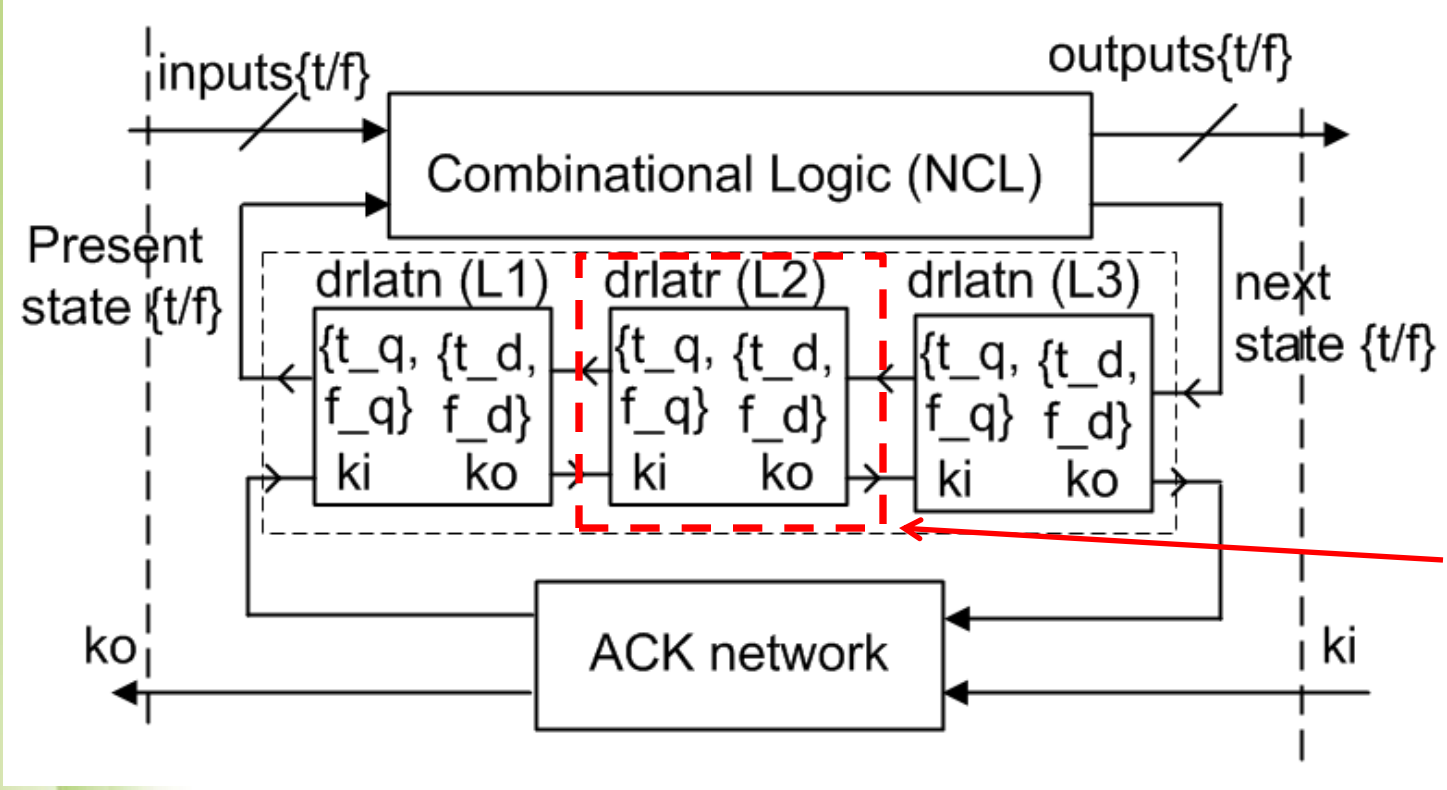




rfd:request for data, *rfn*: request for NULL

Data-driven design with data arrival, acknowledgements controlling the data flow; external ports active every compute cycle.

Finite State Machine



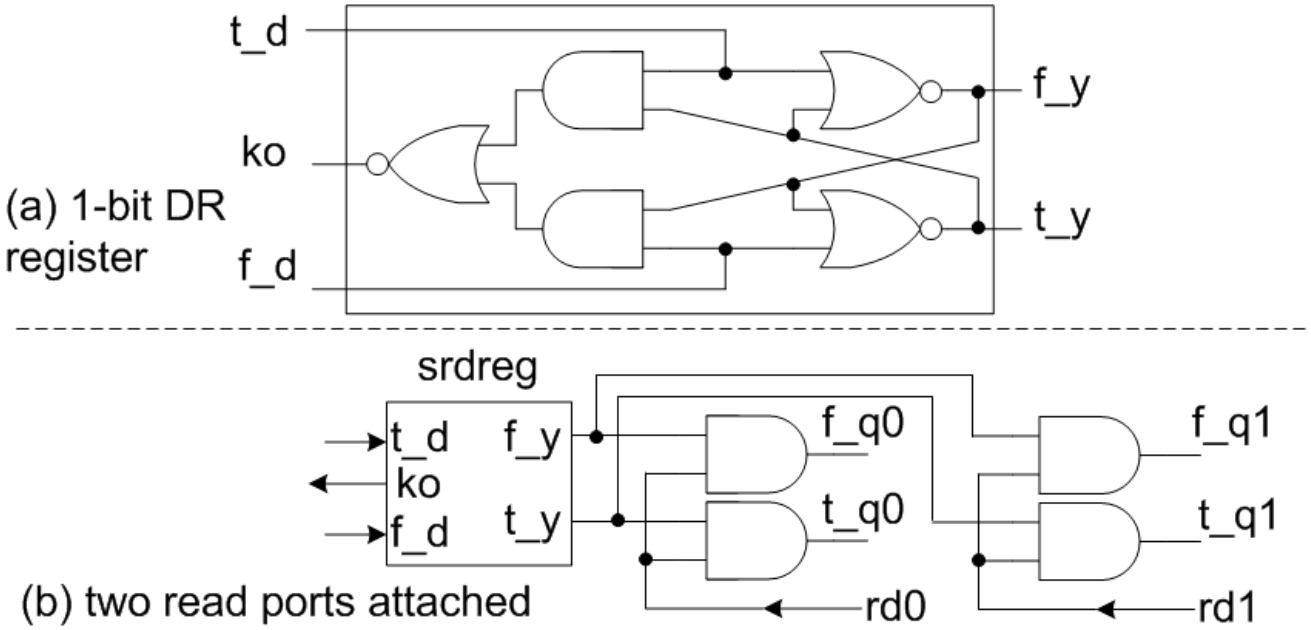
Must be reset to Data (either Data-0 or Data-1) to insert token on ring.

Three-half latches used for registers involved in a loop with middle half-latch having initial data at reset.

Data-driven design in that all logic is dual-rail, no separation of control/datapath, external ports are active every compute cycle.

NCL Systems using Balsa

Balsa [Bardsley, Univ. of Manchester '98] is a well-known asynchronous synthesis system that can generate designs that can use NCL for combinational logic blocks (supports other logic styles as well). Registers/control do not use NCL.



Very efficient from a transistor viewpoint.

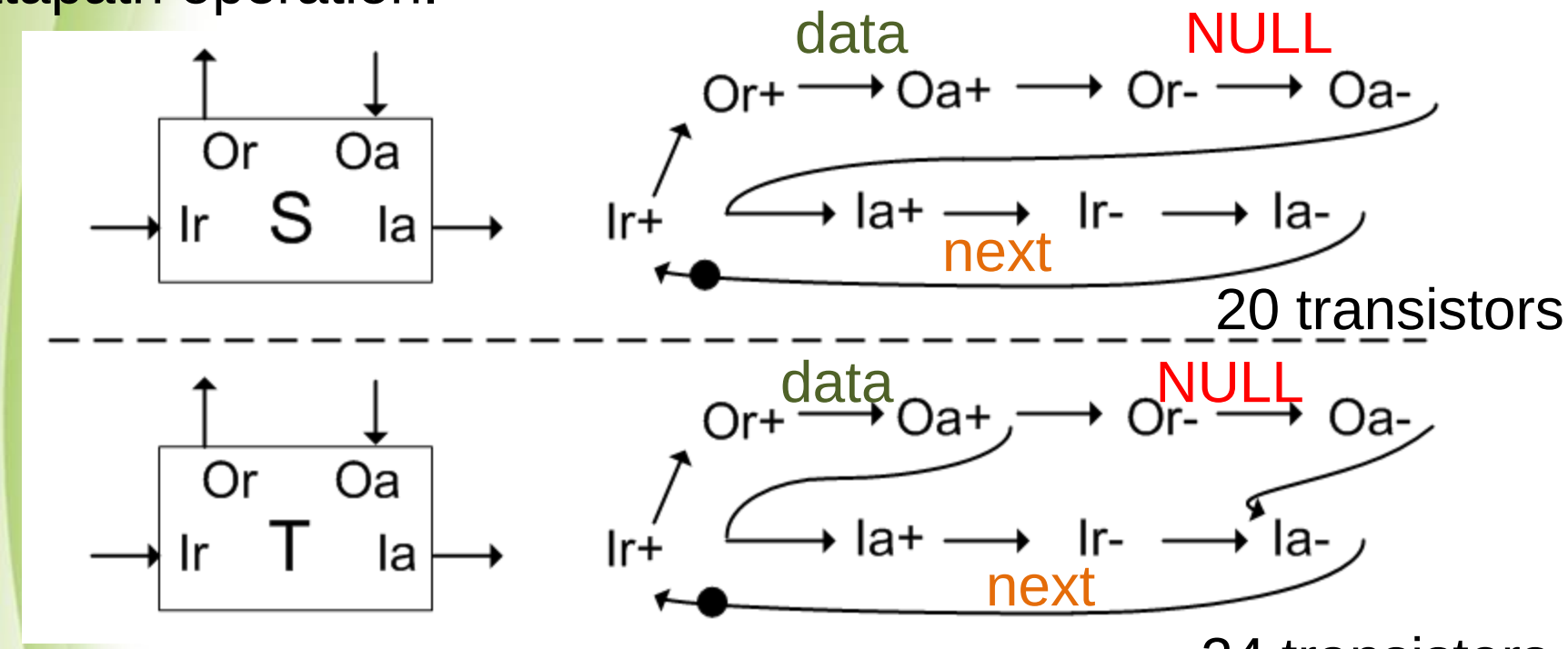
Read ports give conditional access to data.

This register has a low-true ackout (ko)

NCL Combinational logic: Balsa uses dual-rail expanded primitive gates + optimized complex gates (full-adder, others)

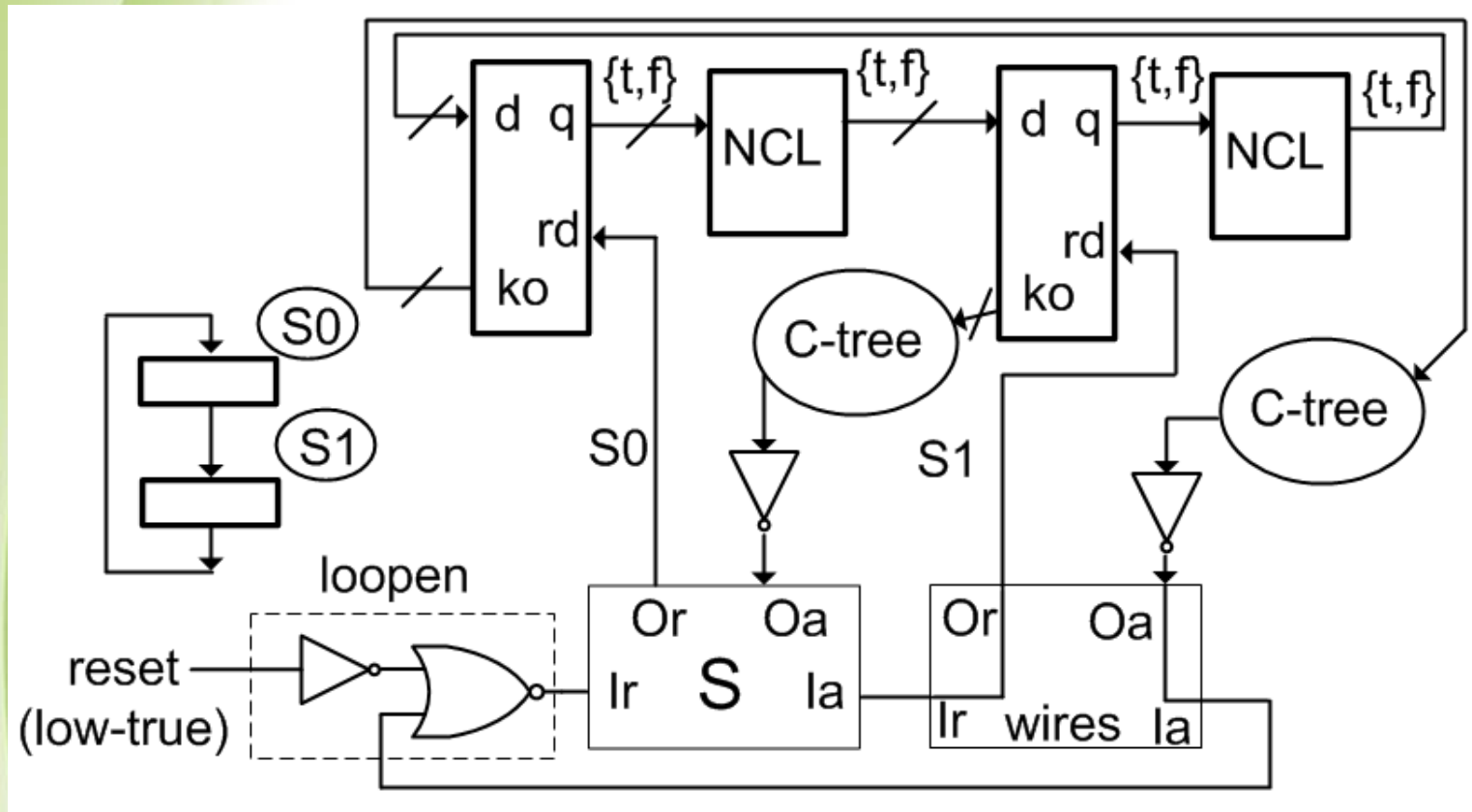
Balsa-style Control

Balsa control uses single-rail handshaking elements (S-element, T-element) to implement sequencers that control datapath operation.



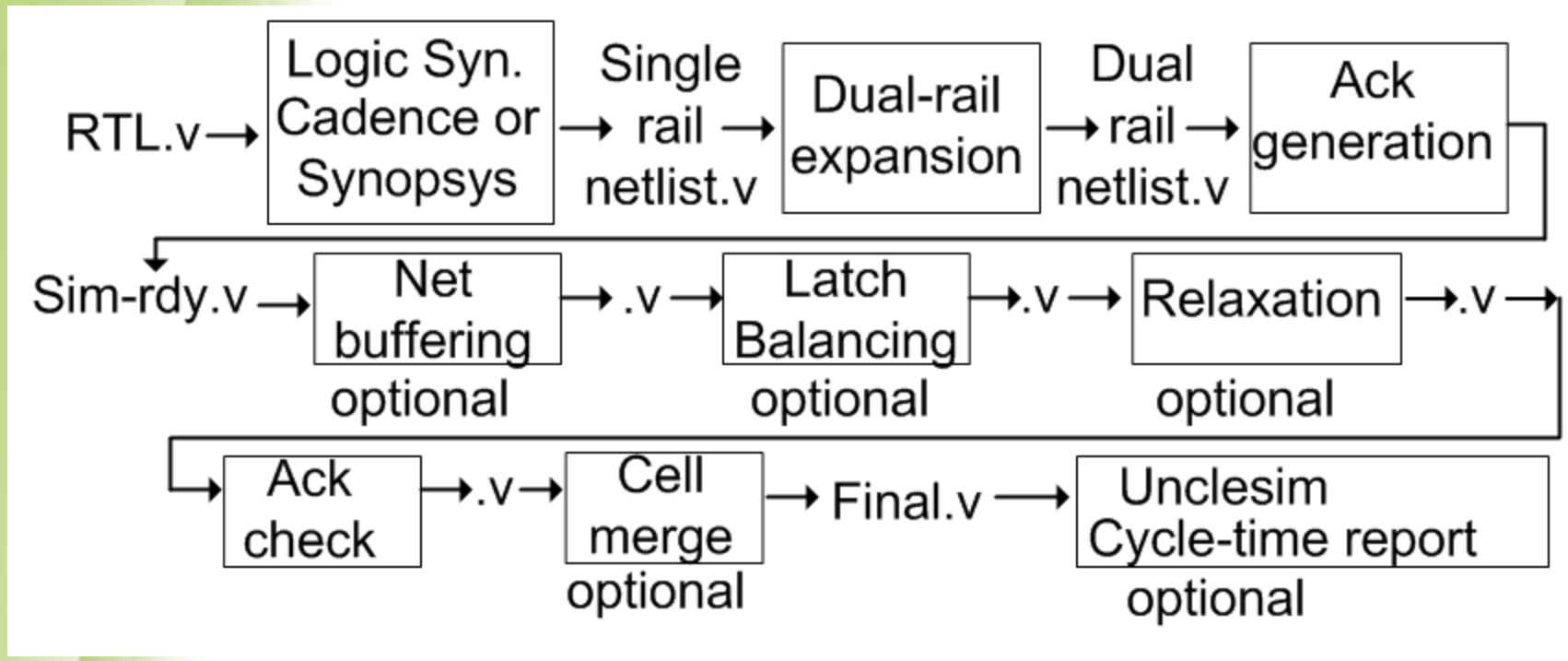
T-element offers more currency than S-element (Oa return to null overlapped with next operation (Ia+).

Example Balsa Datapath/Control



Control is single-rail, datapath is dual-rail. More complex sequencers with choice, conditional looping also possible.

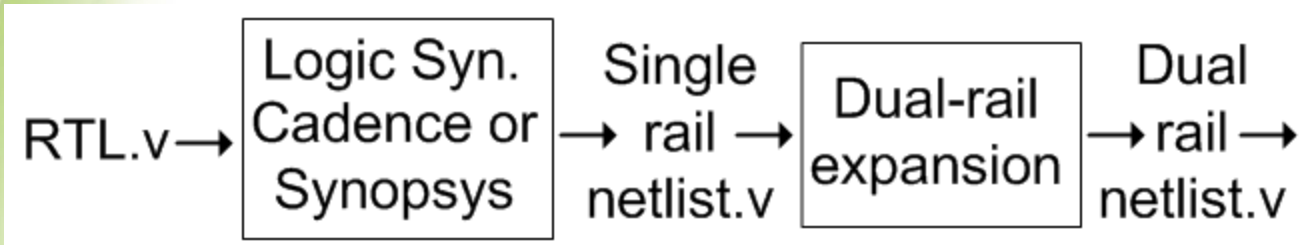
Unified‡ NCL Environment (Uncle)



Both data-driven register/control and Balsa-style register/control (control-driven) is supported (designs can mix the styles).

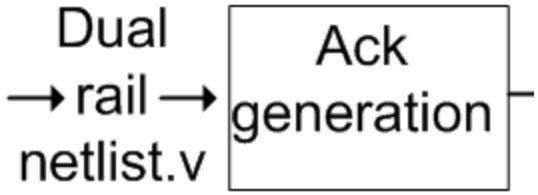
‡ Somewhat pretentious, not yet fully realized and may never be.

RTL to Single-rail to Dual-rail



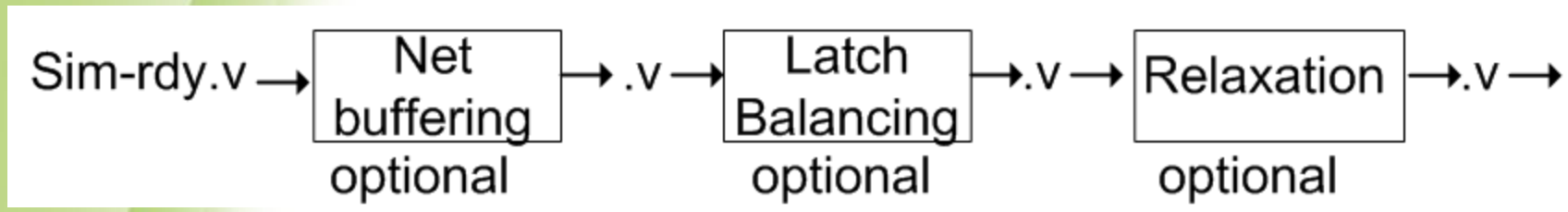
- Area-driven RTL synthesis, weak linkage between timing in .lib and final design, needs to be improved.
- Single-rail netlist output file contains:
  - Primitive gates (AND2, OR2, XOR2, NOT, D-latch, DFF), complex gates (MUX2, FULL ADDER) that are inferred from RTL statements by synthesis.
  - Black-box gates generated from parameterized modules supplied in Uncle that implement various asynchronous functions such as Balsa-style registers, control; specialized functions (arbiter, merge gates)

Ack Generation



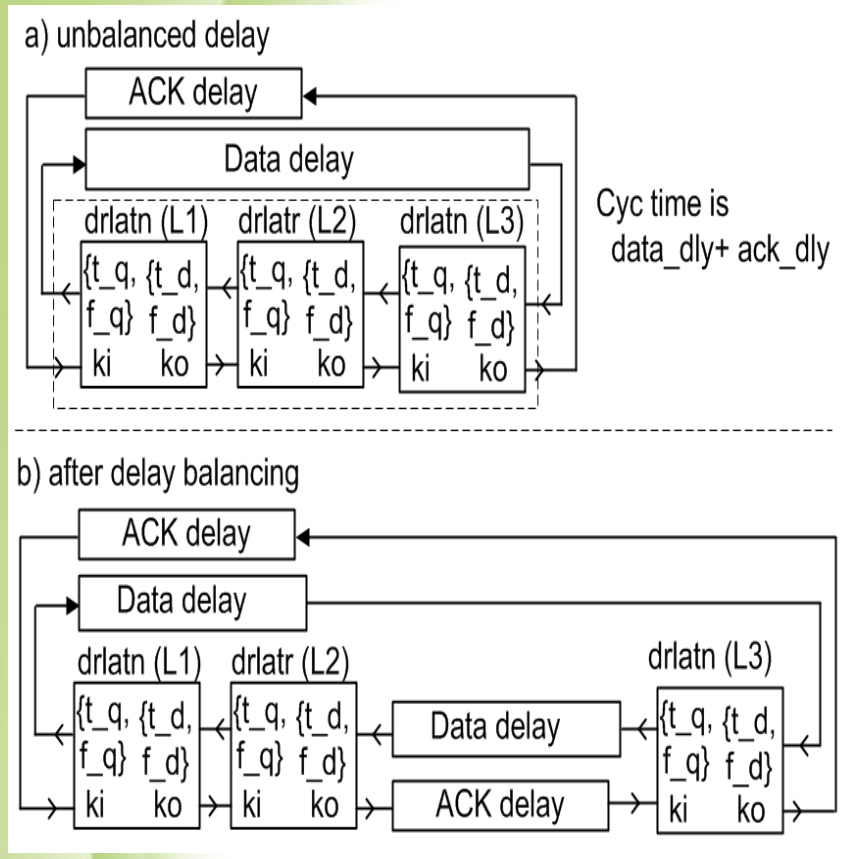
- Ack generation is area-driven and ensures that all data sources receive acks from data destinations
  - Ack networks for latches with common destinations are merged; common cgate sub-trees across different acks are factored and shared
- An ack checker step is included at the end of the flow to check ack network validity
  - Sanity check to ensure intermediate optimization steps have not broken the ack network.

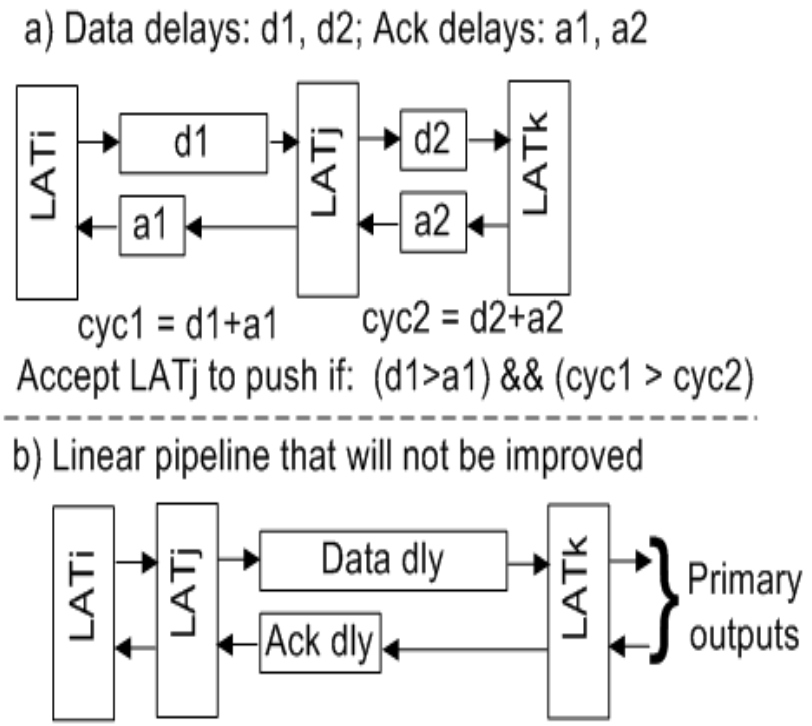
Optimizations



- Net buffering: buffers nets to meet user-specified maximum transition time
  - Timing data uses non-linear delay model (NLDM) – two-axis tables use input transition time, output load. NLDM data from 65 nm technology based on pre-layout transistor models. Library had four inverter variants, three AND2 variants, two register variants, and two variants of most commonly used NCL gates.
- Latch balancing – pushes half-latches to improve performance
- Relaxation – area optimization to reduce gate count of NCL dual-rail expanded logic (Cheoljoo/Nowick Async'2008).

Latch Balancing Details





- Logic pushed across latch boundaries to reduce data+ack cycle time
- Iterative algorithm; multiple candidate latches pushed one gate level each iteration
- Algorithm halts when no cycle time improvement found.

Feature Comparison

	Balsa	Uncle	ATN (Cheoljoo/Nowick)
Combinational synthesis	yes	yes	yes
Control synthesis	yes	Data-driven only (control-driven manual instantiation)	no
Logic Style	Different dual-rail styles, bundled data	NCL only	NCL only
Behavioral simulation	yes	limited	limited
Area optimizations	no	Relaxation, limited cell merging, ack sharing	Relaxation, cell merging
Performance optimizations	Language features allow area, perf. tradeoffs by coding style	RTL style allow area/perf. tradeoffs, latch balancing, net buffering	Timing-driven relaxation
Timing model	Fixed delay	NLDM	Fixed delay

Uncle vs. Balsa Design Comparison Methodology

- Used designs for which published Balsa code was available
  - Balsa code that was used was written in a high performance style
- Designs mapped to same gate level library for apples-to-apples comparison
  - Designs verified at both gate and transistor levels
  - Transistor simulation used pre-layout transistor models in 65 nm technology; Cadence Ultrasim used for verification.
  - All test benches were self-checking

Design Example: 16-bit Integer GCD

Uncle versions

Uncle ver.	DD	DD/NB	DD/LB/ NB	CD	CD/NB
transistors	16192	16226	20128	8658	8662
*	1.87	1.87	2.32	1.00	1.00
cyc. time (ns)	105.7	86.0	64.9	75.7	62.4
*	1.69	1.38	1.04	1.21	1.00
energy (pJ)	32.4	35.3	49.7	10.2	10.8
*	3.17	3.44	4.85	1.00	1.05

Conditional port activity caused data-driven designs to be large, slow. Latch balancing helped DD performance. Control driven produced best results.

DD: data-driven; NB: net-buffered; LB: latch-balanced, CD: control-driven
Note: Control-driven == Balsa style registers/control

Design Example: 16-bit Integer GCD

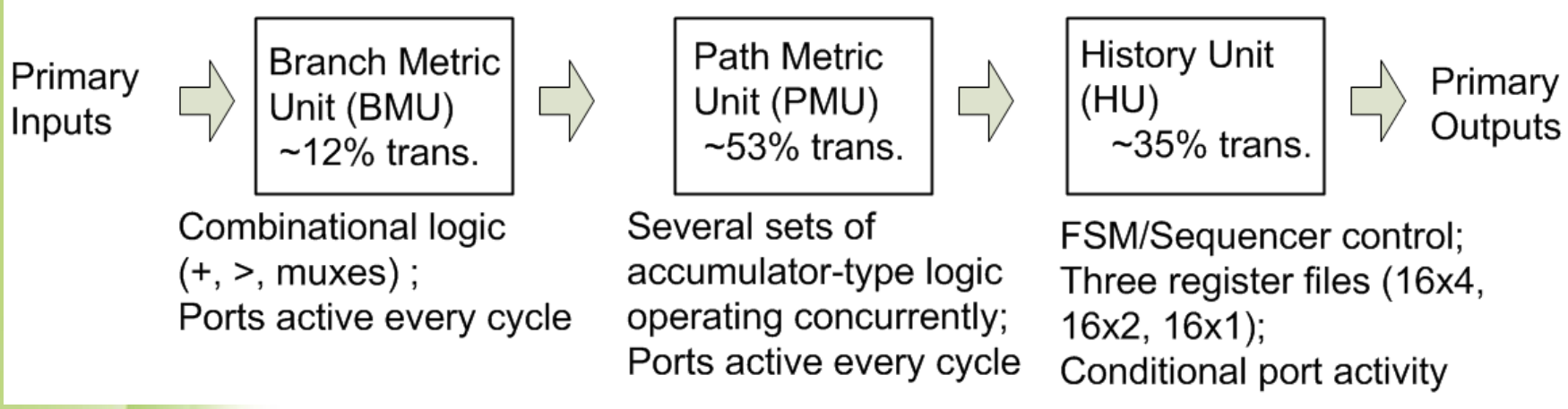
Uncle vs. Balsa

	transistors		Cyc time (ns)		Energy (pJ)	
	Balsa	Uncle (CD/ NB)	Balsa	Uncle (CD/ NB)	Balsa	Uncle (CD/ NB)
	11455	8662	85.2	62.4	13.7	10.8
RTB	1.32	1.00	1.37	1.00	1.27	1.00

Balsa used more read ports on registers reducing loading but increasing transistor count. Net buffering helped offset increased loading in Uncle design, improved performance.

RTB: ratio-to-best; DD: data-driven; NB: net-buffered; LB: latch-balanced, CD: control-driven

Viterbi Decoder



- Balsa code from published source (written for high performance) [L. T. Duarte PhD diss., 2010, Univ. Manchester]
- Investigated different Uncle versions for each block
  - Compared best Uncle vs. Balsa for each block
- Final Balsa/Uncle versions ran complete code (each multiple modules) in one pass through synthesis systems to get final netlists.
  - Both verified at gate and transistor levels with same vectors.

Branch Metric Unit: Uncle vs. Balsa

	transistors		Cycle time (ns)		Energy (pJ)	
	Balsa	Uncle (DD/NB)	Balsa	Uncle (DD/ NB)	Balsa	Uncle (DD/NB)
	9040	5338	9.30	8.87	2.33	1.35
RTB	1.69	1.00	1.05	1.00	1.73	1.00

- Uncle version just combinational logic with half-latch on output
- Balsa version used loop splitting to split combinational logic into concurrent blocks that increased parallelism of internal computations at the cost of more transistors.
  - Has overhead of more transistors

RTB: ratio-to-best; DD: data-driven; NB: net-buffered;

Path Metric Unit: Uncle Versions

Uncle ver.	DD/NB	DD/NB/LB	DD/NB/LB+	CD/NB
transistors	20184	21778	24561	18838
RTB	1.07	1.16	1.30	1.00
cyc. time (ns)	13.4	13.4	6.9	13.3
RTB	1.93	1.93	1.00	1.91
energy (pJ)	5.1	5.7	6.8	4.6
RTB	1.12	1.24	1.48	1.00

- Latch balancing did not improve data-driven performance until extra half-latch stage added on primary outputs to give more latch movement freedom; data-driven had highest performance.
- Control-driven approach used fewest transistors as expected.

RTB: ratio-to-best; DD: data-driven; NB: net-buffered; LB: latch-balanced, LB+: latch-balanced, extra latch stage on primary outputs CD: control-driven

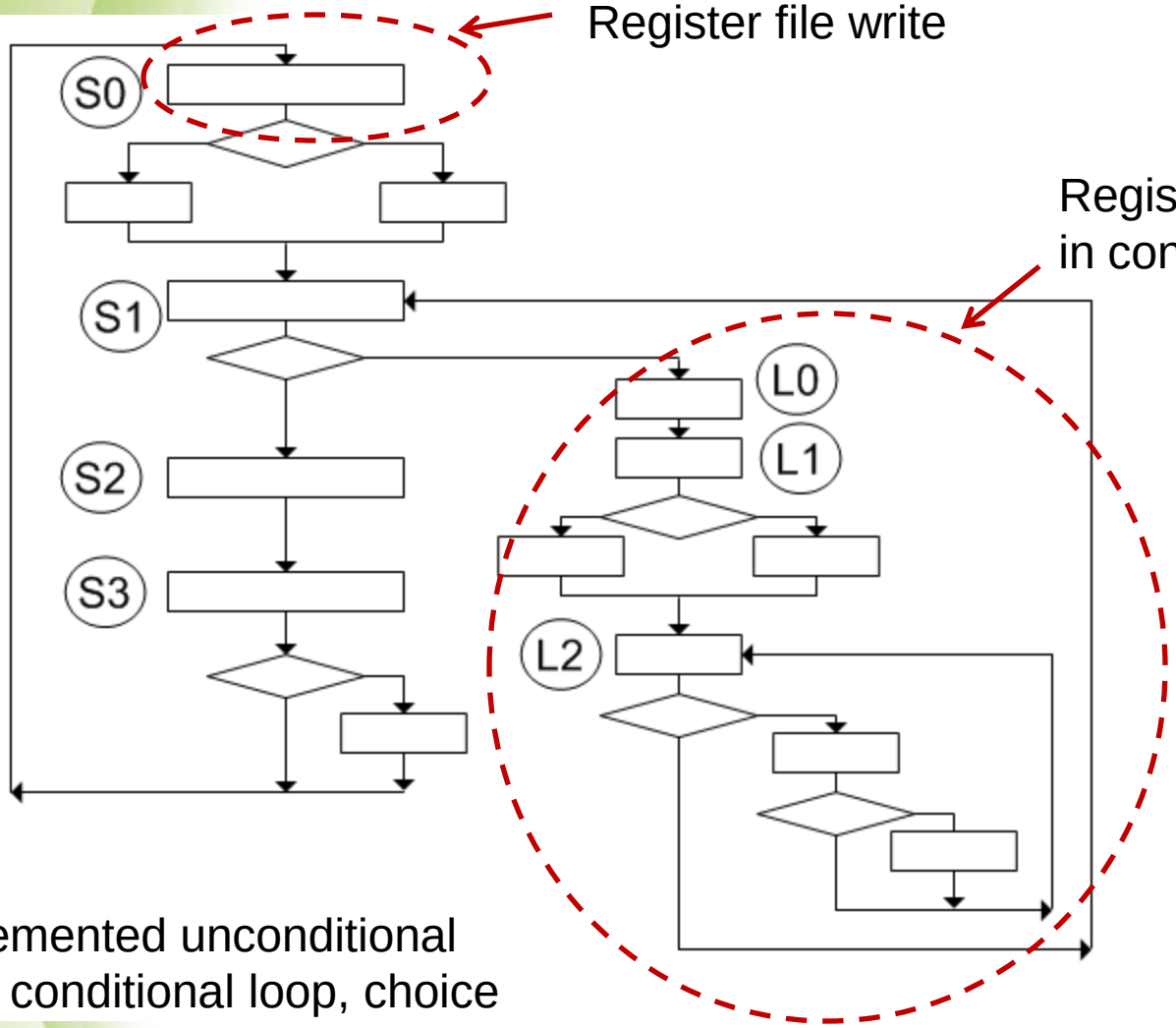
Path Metric Unit: Uncle vs Balsa

	transistors		Cycle time (ns)		Energy (pJ)	
	Balsa	Uncle (DD/ NB/LB+)	Balsa	Uncle (DD/ NB/LB+)	Balsa	Uncle (DD/NB/ LB+)
	38328	24561	9.39	6.94	9.73	6.81
RTB	1.56	1.00	1.35	1.00	1.43	1.00

- Uncle data-driven approach with latch balancing, net buffering compares favorably in all areas to Balsa version
  - Without latch balancing, Uncle implementation would have been slower.
  - Balsa implementation was faster than Uncle's control-driven implementation; Balsa has some performance enhancement features not currently implemented in Uncle.
  - Transistor discrepancy between Balsa and Uncle appears to be mostly in the trellis sub-module which is simply wires in Uncle, but channels with enclosure logic in Balsa.

RTB: ratio-to-best; DD: data-driven; NB: net-buffered; LB: latch-balanced, LB+: latch-balanced, extra latch stage on primary outputs CD: control-driven

History Unit Control



Control optimization was implemented that overlapped register file write return-to-NULL with S2/S3 only if conditional loop (L0….) was not executed.

Implemented unconditional loop, conditional loop, choice

History Unit: Uncle vs Balsa

		Balsa	Uncle CD/NB	Uncle CD
	transistors	21819	16471	16425
	RTB	1.33	1.00	1.00
v1	cyc. time (ns)	10.8	6.8	8.4
	RTB	1.60	1.00	1.25
	energy (pJ)	1.34	1.17	1.07
	RTB	1.26	1.09	1.00
v2	cyc. time (ns)	230.7	161.3	192.0
	RTB	1.43	1.00	1.19
	energy (pJ)	25.4	19.6	18.7
	RTB	1.36	1.05	1.00

V1: no internal-loop execution
V2: internal loop execution

Control optimization for ‘V1’ set in Uncle implementation provided performance boost.

Unclear as to exact reason for performance boost on ‘V2’ set (could be a mixture of control + datapath efficiency).

RTB: ratio-to-best; CD: control-driven; NB: net-buffered;

Viterbi Decoder: Uncle vs. Balsa

	transistors		Cycle time (ns)		Energy (pJ)	
	Balsa	Uncle	Balsa	Uncle	Balsa	Uncle
	71370	46752	22.0	17.3	15.0	10.5
RTB	1.53	1.00	1.27	1.00	1.43	1.00

- Transistor counts in this table does not match sums of previous tables since entire source processed at one time through respective tools
  - Balsa's transistor count is ~4% higher than published source.

RTB: ratio-to-best; CD: control-driven; NB: net-buffered;

Observations/Conclusions

- Uncle's RTL approach requires more effort by the designer than Balsa's approach, especially for control-driven modules
  - But can result in a higher quality design
- Latch balancing is a performance win for data-driven designs with always active ports
- Data-driven style better for modules with always active ports if performance is goal.
- Control-driven style (Balsa-style registers/control) better for modules with conditional port activity.

Future Work/ Paper Contributions

- Future work
  - Direct NCL synthesis with input completeness (M. Thornton)
  - Support for multi-threshold NCL with sleep (S. Smith)
  - Timing-driven ack-generation, timing-driven relaxation
  - Net-buffering for critical paths, wire load model
  - Automated half-latch insertion for performance
  - Better timing connection between input synthesis library and final gate level netlist
- Paper contributions:
  - Demonstration of asynchronous RTL methodology (again…)
  - Latch balancing optimization
  - Design data point for future comparison

Thanks for listening!
Questions?

Uncle available at sites.google.com/site/asynctools
Automated regression testing for all designs, user manual.
Source available on request.

Reviewer Questions

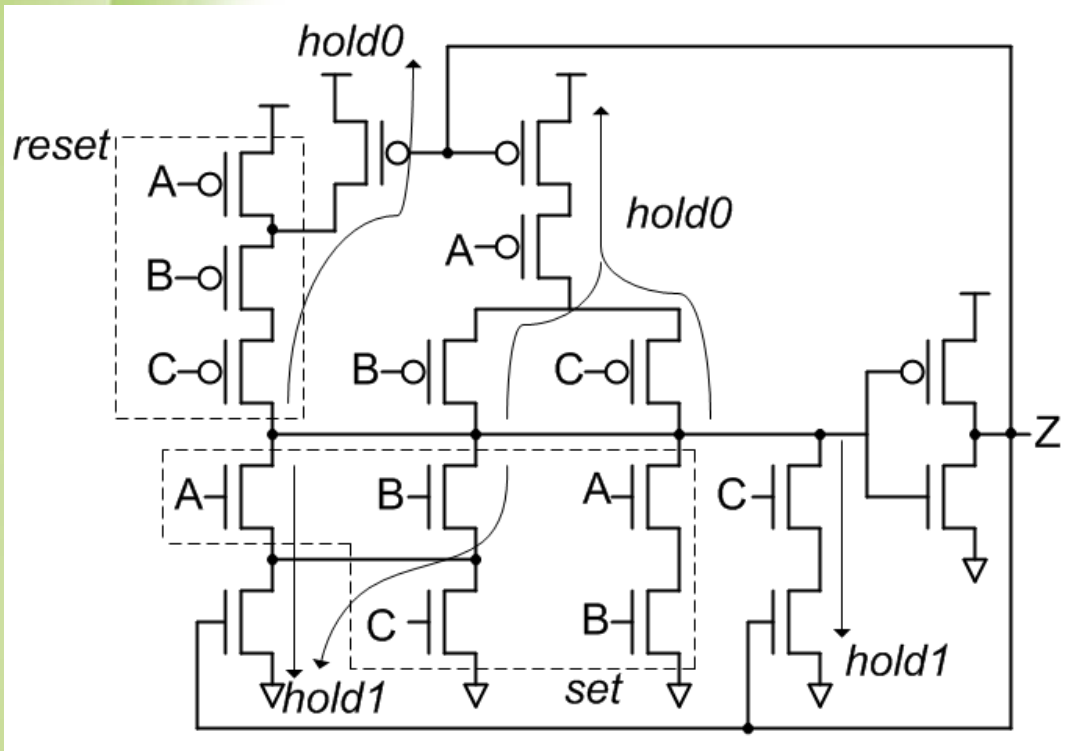
- Why was iterative algorithm that only pushed one gate level used for latch balancing instead of a standard retiming algorithm for optimum latch location?
  - It was not used because of difficulties in predicting new ack network performance, since ack network changes based on where latches are located in logic. It is acknowledged that a standard retiming algorithm would give a better starting point and save CPU time, unclear if result quality would be better.
- Why use unit delays for gates in the Synopsys/Cadence library use for synthesis?
  - This is an acknowledged weakness – delays closer to the actual dual-expanded gate delays should be used (the NLDM timing models for gates were done late in project, did not make it into Synopsys/Cadence library).
- Why did net buffering ignore wire loading?
  - It is acknowledged that a wire load model needs to be added.
- Where do the black-box gate, parameterized modules come from?
  - They are provided in the Uncle release. User has freedom to add new parameterized modules if desired.

Static CMOS Implementation

- Static CMOS NCL gate has reset, set, hold0, hold1 blocks

$Z = set + (Z^{-} \bullet hold1)$; Z^{-} prev output

$Z' = reset + (Z'^{-} \bullet hold0)$;



TH23

set = AB + AC + AB

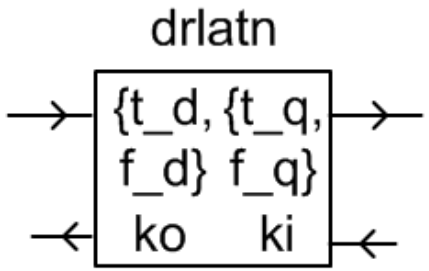
reset = A' B' C'

hold0 = set'
=A'B' +A'C' + B'C'

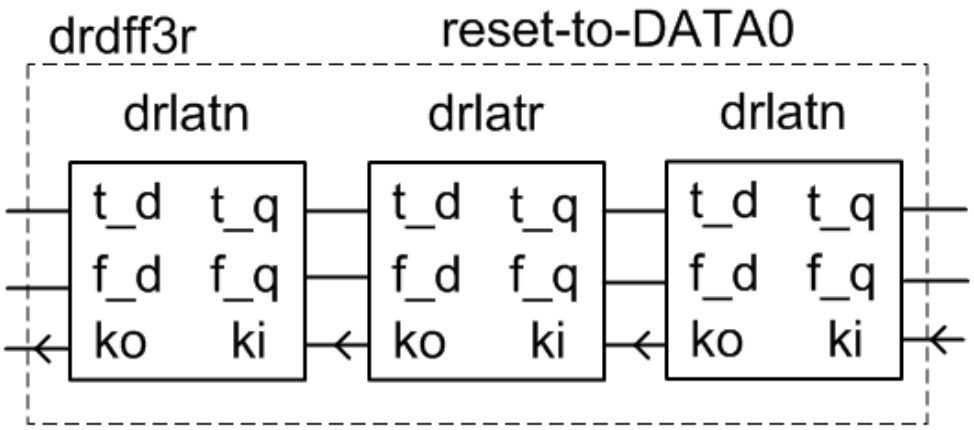
hold1 = (inputs or'ed)
= A + B + C

RTL Example Snippets

(a)
```
always @* begin
   if (clk == 1) q <= d;
end
```



Clocked D-latch maps to dual-rail half-latch during dual-rail expansion.



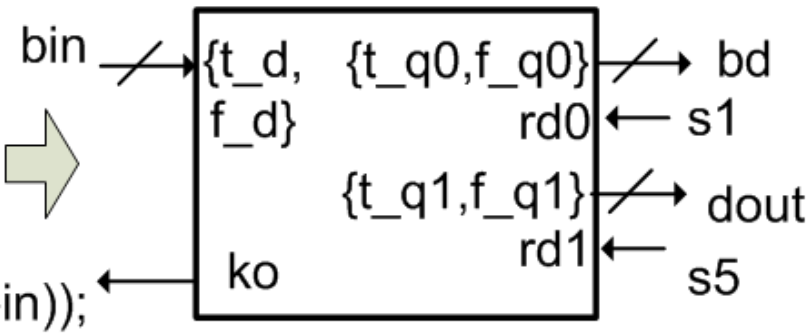
(b)
```
always @ (negedge reset or posedge clk)
begin
   if (reset == 0) q <= 0;
   else q <= d;
end
```

Clocked DFF maps to three half-latch structure with initial data in middle latch during dual-rail expansion.

RTL Example Snippets (cont.)

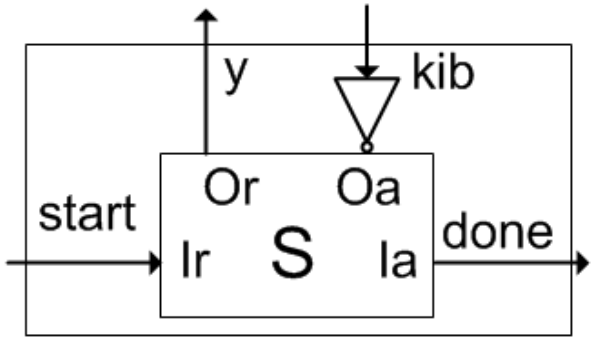
(a) SR-latch register with two read ports

```
`define WIDTH 16
 wire  [(`WIDTH-1):0] bd,dout, din;
 wire  s1, s5;
dreg2port_n #(.WIDTH(`WIDTH))
       b_master (.q0(bd),.rd0(s1),
                 .q1(dout),.rd1(s5),.d(bin));
```



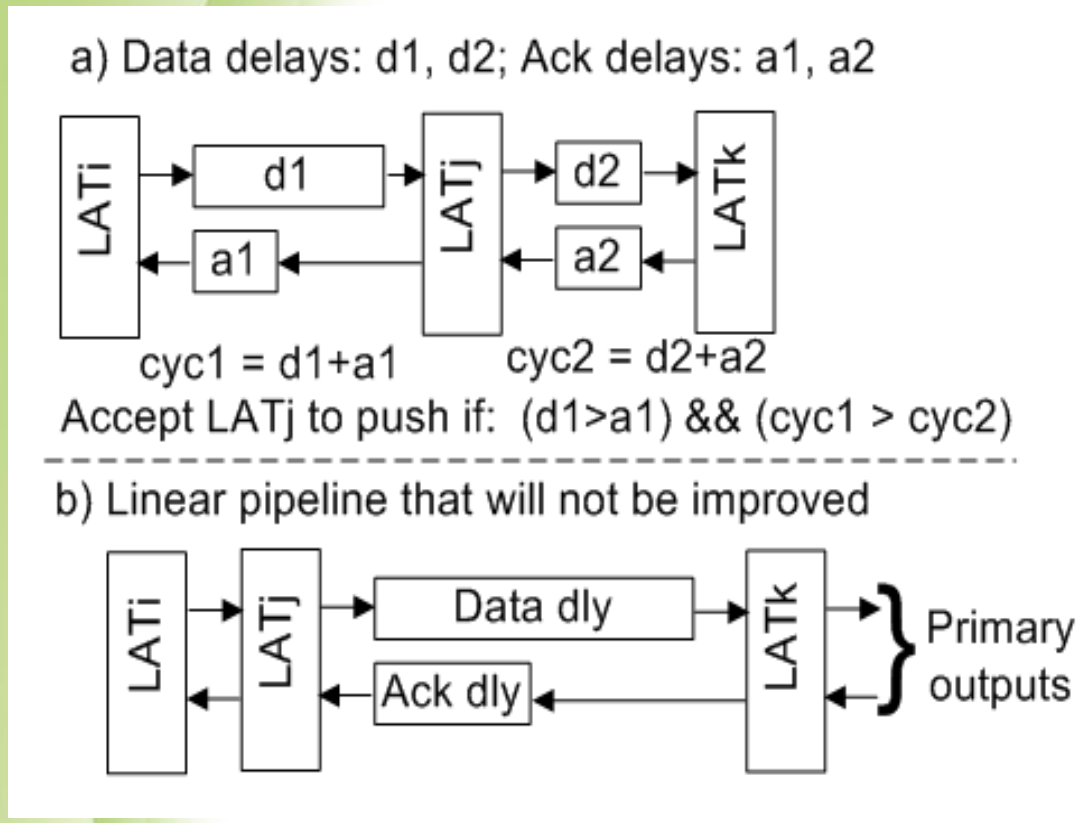
(b) S-element instantiation

```
 wire s0_start,loop_start, s0, log0;
assign log0 = 0;  //supress lint
seqelem_kib  u1
  (.start(s0_start), .done(loop_start),
   .y(s0), .kib(log0));
```



Parameterized modules are used to implement functionality that cannot be inferred from RTL. These expand to black-box gates ignored by synthesis and passed to the gate-level file.

Latch Balancing Algorithm



Caveat: Current algorithm will not find improvement in (b) even though improvement exists.

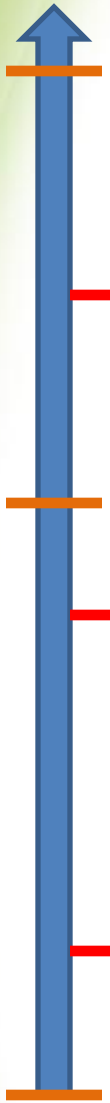
Iterative algorithm that pushes candidate latches by one gate level.

Latches pushed in only one direction (LATj towards LATi).

Latch candidates are identified using several sorting/pruning stages to identify those most likely to improve performance.

Algorithm halts when no further improvement made. Delays calculated using NLDM timing data.

Feature Comparison



Behavioral Synthesis

Balsa: complete system from Balsa spec, simulation of Balsa spec, control synthesis, fixed delay timing model, user can control area/performance via language constructs, can produce bundled data, different dual-rail logic styles.

Modern RTL flows

Uncle: complete system from Verilog RTL, limited RTL simulation, control synthesis only for data-driven approach, Balsa style reg/control via parameterized macros, NLDM timing, latch balancing netlist optimization for performance, area-driven relaxation

ATN [Jeong/Nowick]: combinational only from Blif/Verilog gate netlist, timing/area-driven relaxation, technology mapping, fixed delay timing model.

Manual Netlisting