Part II. MINIMALIST: HANDS-ON TUTORIAL

(Release v2.0)

Steven M. Nowick
Columbia University
(nowick@cs.columbia.edu)

November 24, 2007

MINIMALIST: Funding Acknowledgments

v2.0 Release (NEW):

This work was supported by NSF ITR Award No. NSF-CCR-0086036.

v0.9 and v1.0-v1.2 Releases (1994-2001):

This work was supported by NSF Award Nos. NSF-CCR-99-88241, NSF-MIP-95-01880 and NSF-MIP-9308810; a research grant from IBM Corporation; and an Alfred P. Sloan Research Fellowship.

MINIMALIST v2.0: Download Site

Available as part of the "CaSCADE" async tool package

Download site:

http://www1.cs.columbia.edu/~nowick/asynctools

Current v2.0 support:

Linux only*

*(earlier Solaris versions not actively supported or updated)

Supporting material:

complete tutorial: in 2 parts ['tutorial' directory]

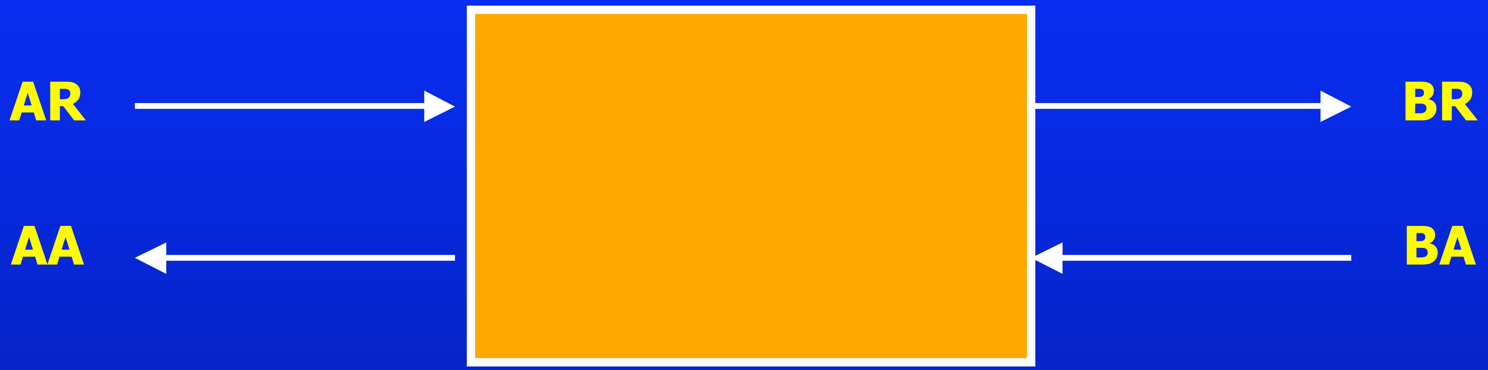
(i) Overview of MINIMALIST

(ii) Hands-On Tutorial (this file)

benchmark examples ['examples' directory]

other documentation: INSTALL, README, etc. ['docs' directory]

EXAMPLE #1: Martin Q-Element



<u>From Caltech:</u>

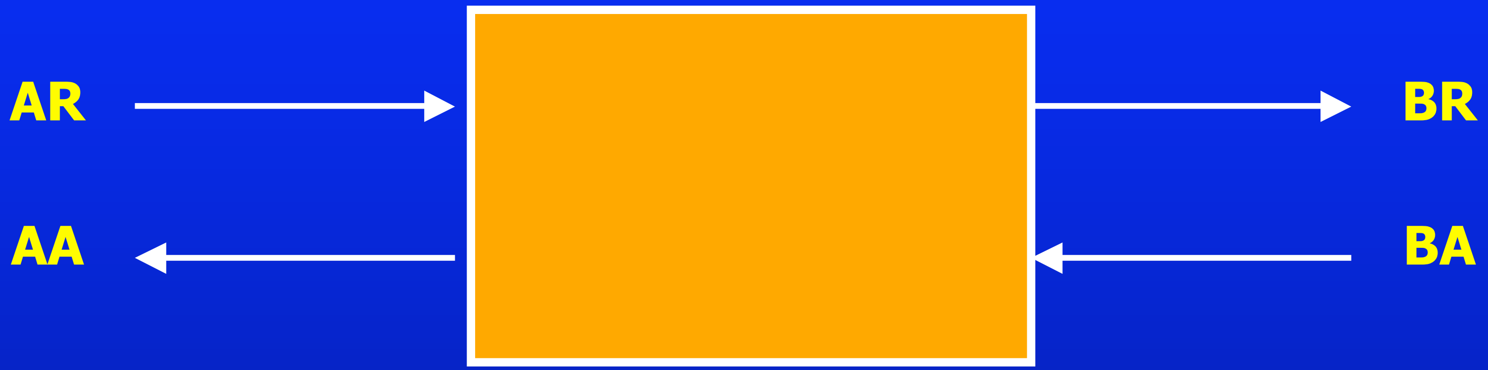
Alain Martin,

"Programming in VLSI: from Communicationg Processes to Delay-Insensitive Circuits".

Chapter in "Developments in Concurrency and Communication",

(ed. C.A.R. Hoare), Addison Wesley, UT Year of Programming Series, pp. 1-64 (1990).

EXAMPLE #1: Martin Q-Element

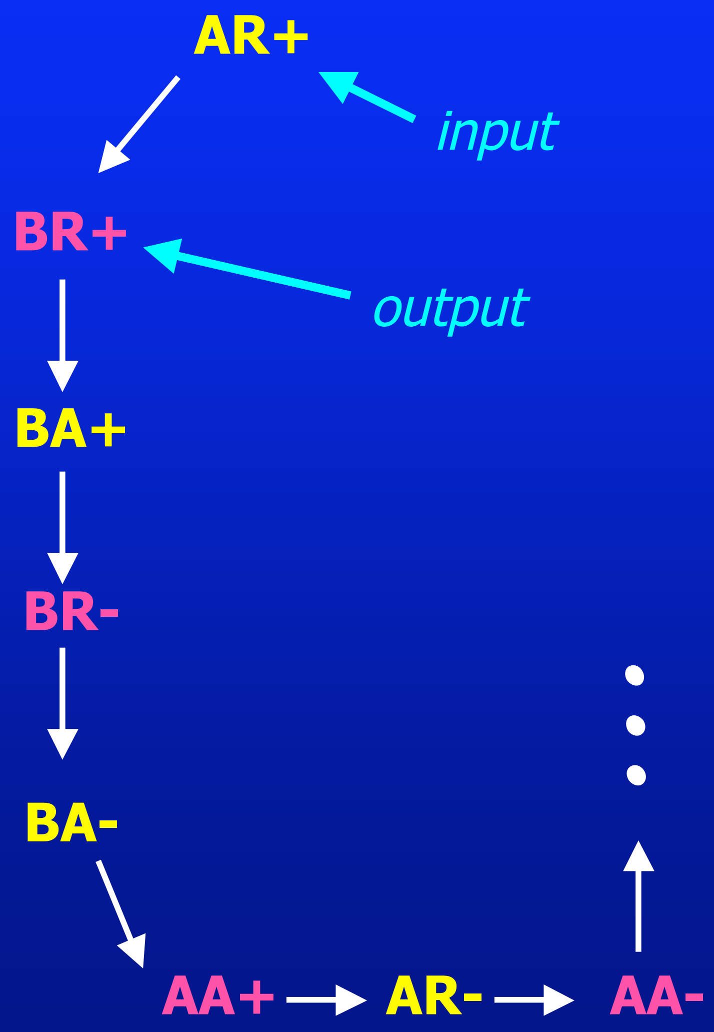


Handshaking Protocol:

$$\begin{tikzcd}
AR+ \arrow[dr] & & & & AA+ \arrow[r] & AR- \arrow[r] & AA- \\
& BR+ \arrow[r] & BA+ \arrow[r] & BR- \arrow[r] & BA- \arrow[u]
\end{tikzcd}$$

Example #1: Martin Q-Element

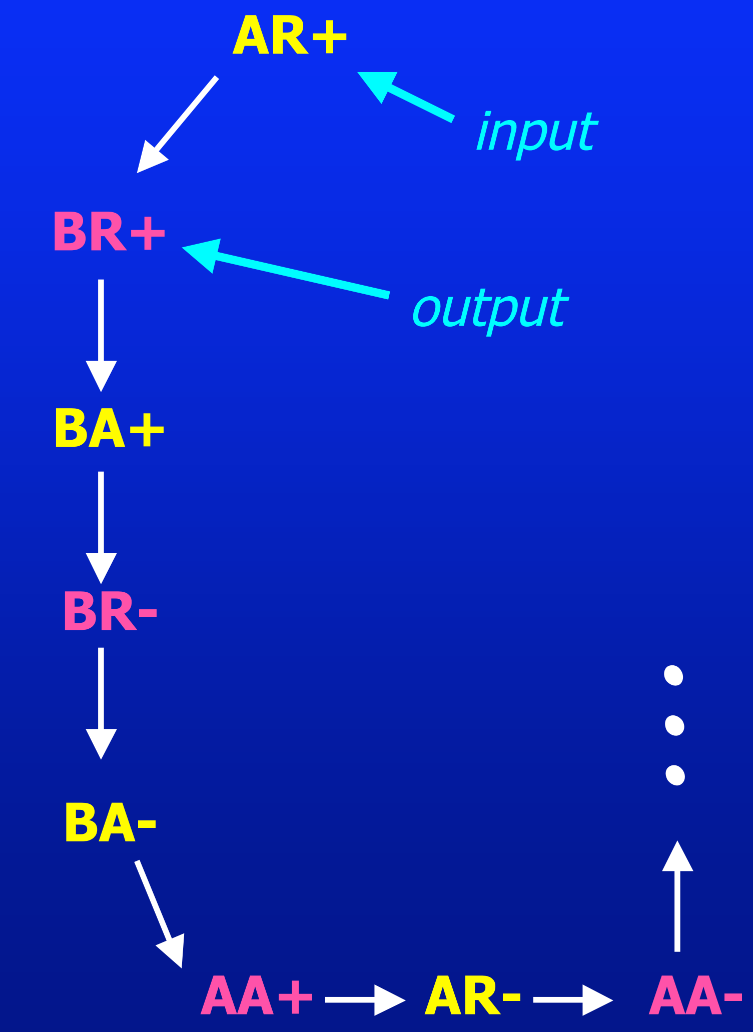
Handshaking Protocol:



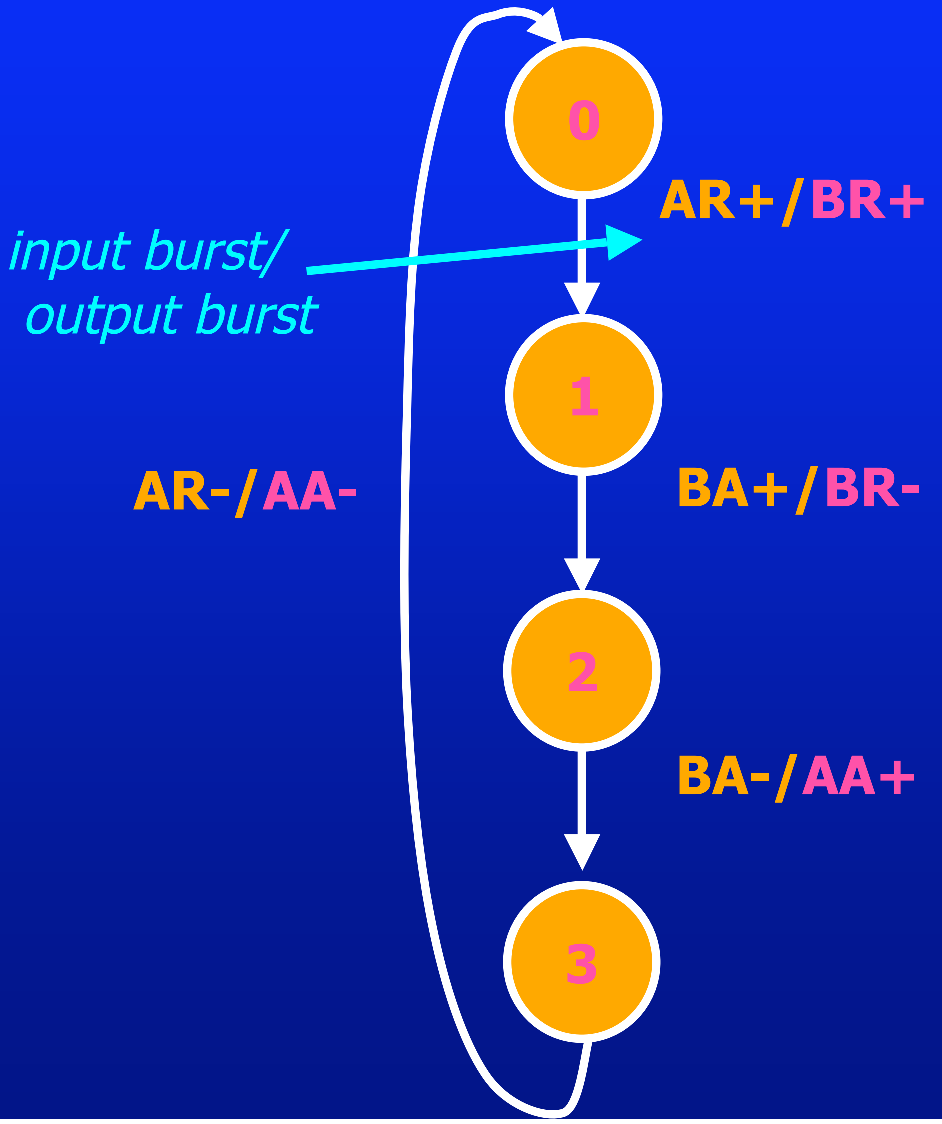
Burst-Mode Specification:

Example #1: Martin Q-Element

Handshaking Protocol:



Burst-Mode Specification:



Example #1: Martin Q-Element

Running MINIMALIST: the Simple Approach

Step #0. Getting Started ...

– First, follow the set-up instructions in the "INSTALL" file

– Go to the "root directory" ("MINIMALIST") of the unpacked tool (use 'cd')

– Create a new subdirectory there *(or modify steps to create it elsewhere):*

```
mkdir min-demo
```

– Go to "min-demo", create an "ex1" subdirectory, and enter it:

```
cd min-demo
mkdir ex1
cd ex1
```

– Copy the Q-element BM specification file from "examples" into "ex1":

```
cp  ../../examples/martin-q-element/martin-q-element.bms  .
```

Example #1: Martin Q-Element

Step #0. Getting Started (cont.) ...

- Next, to start up the "MINIMALIST" tool, type:

```
> MinShell
```

- MINIMALIST will respond with a new prompt:

```
minimalist>
```

- Type 'help' to see top-level help menu:

```
minimalist> help
```

You are now using "MinShell". This environment allow you to do synthesis runs, display specs + circuits, get online help, and even run many standard Unix commands (cd, cp, mv, ls, pwd, …). In general, you will want to activate "MinShell" at the start of a MINIMALIST synthesis session (though most commands can also be run directly from the Linux shell).

Example #1: Martin Q-Element

Running MINIMALIST: the Simple Approach

Step #1. Show BM Specification

(a) Look at "BMS" text file:

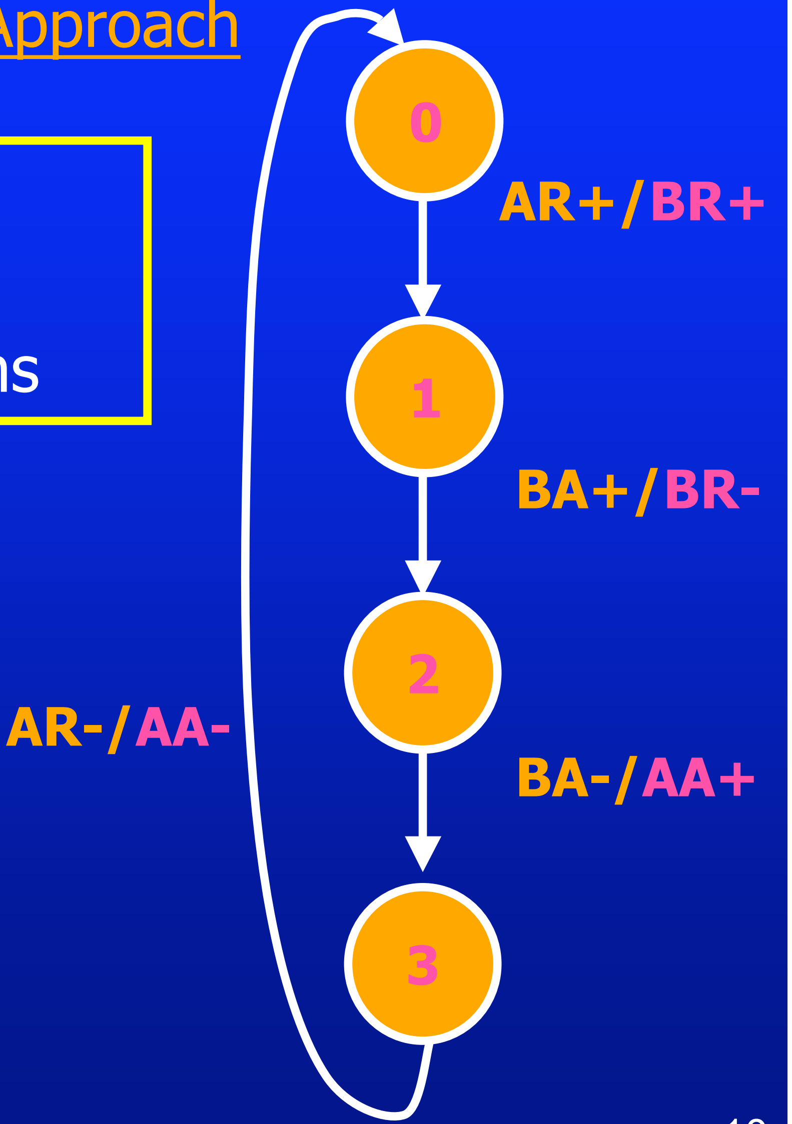
```
> more  martin-q-element.bms
```

```
name martin_q_element

Input    AR  0
Input    BA  0

Output   BR  0
Output   AA  0

0  1   AR+  |  BR+
1  2   BA+  |  BR-
2  3   BA-  |  AA+
3  0   AR-  |  AA-
```



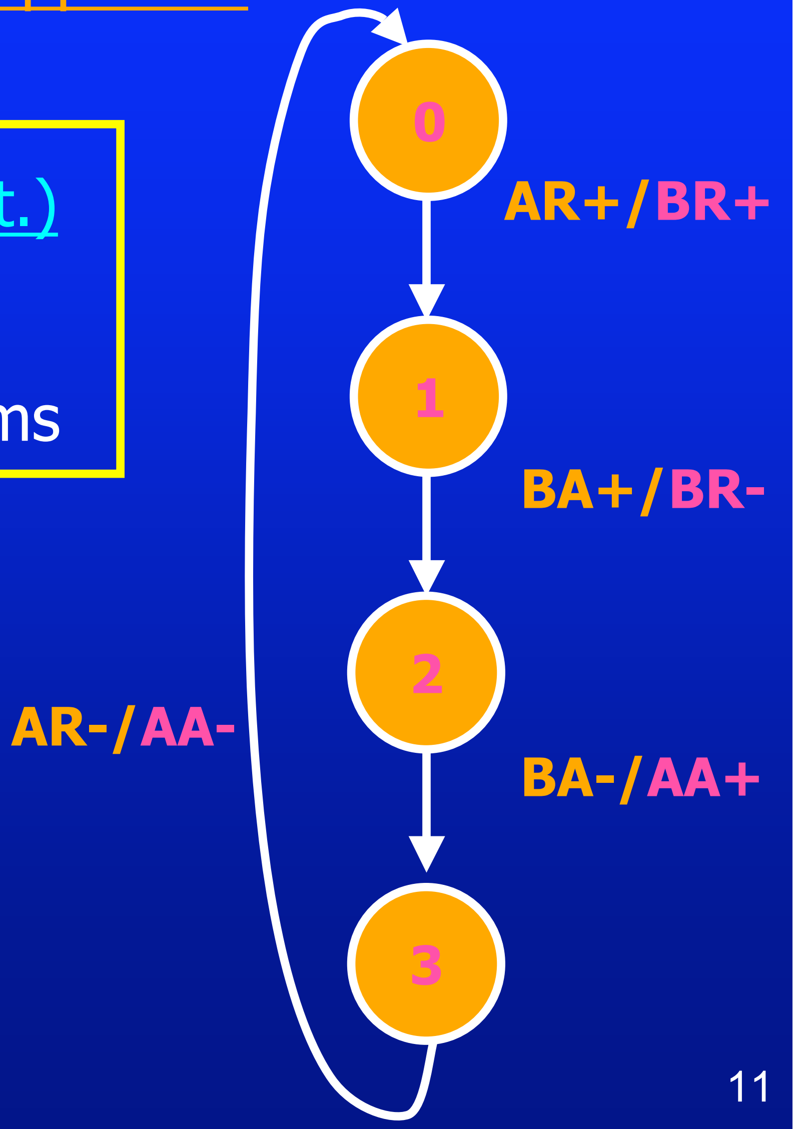
Example #1: Martin Q-Element

Running MINIMALIST: the Simple Approach

Step #1. Show BM Specification (cont.)

(b) Graphic Display:

```
> bms2ps  martin-q-element.bms
```



Example #1: Martin Q-Element

Running MINIMALIST: the Simple Approach

Step #2. Synthesize BM Implementation: using a “script”

```
> minimalist-basic martin-q-element.bms
```

Step #3. Display It:

(a) Text: 2-Level Equations + Results Summary

> [see displayed text output]

(b) Plot NAND/NAND Circuit: [follow displayed instructions:]

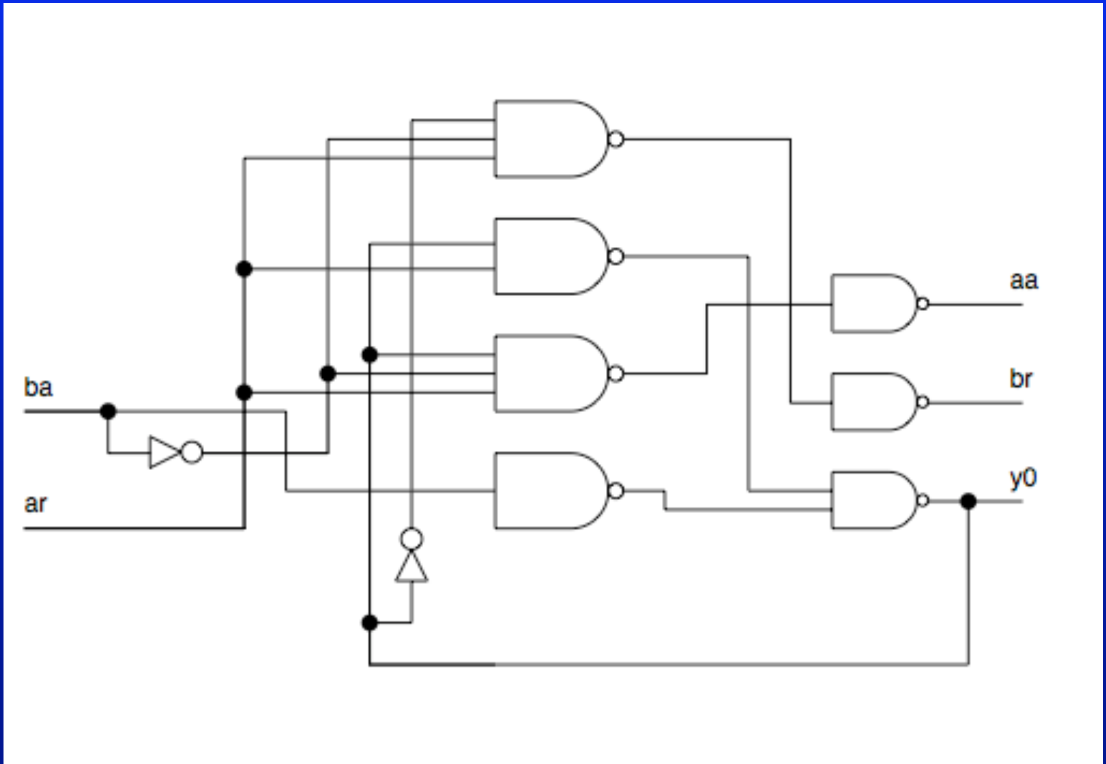
```
> plot_nand martin_q_element-L.sol
```

Example #1: Martin Q-Element

Running MINIMALIST: the Simple Approach

Step #3. Display It (cont.):

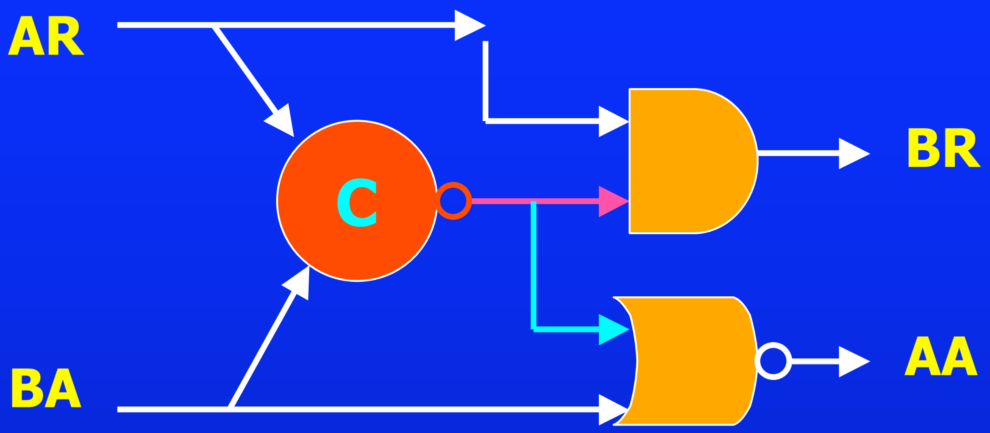
(b) Result of "plot_nand":



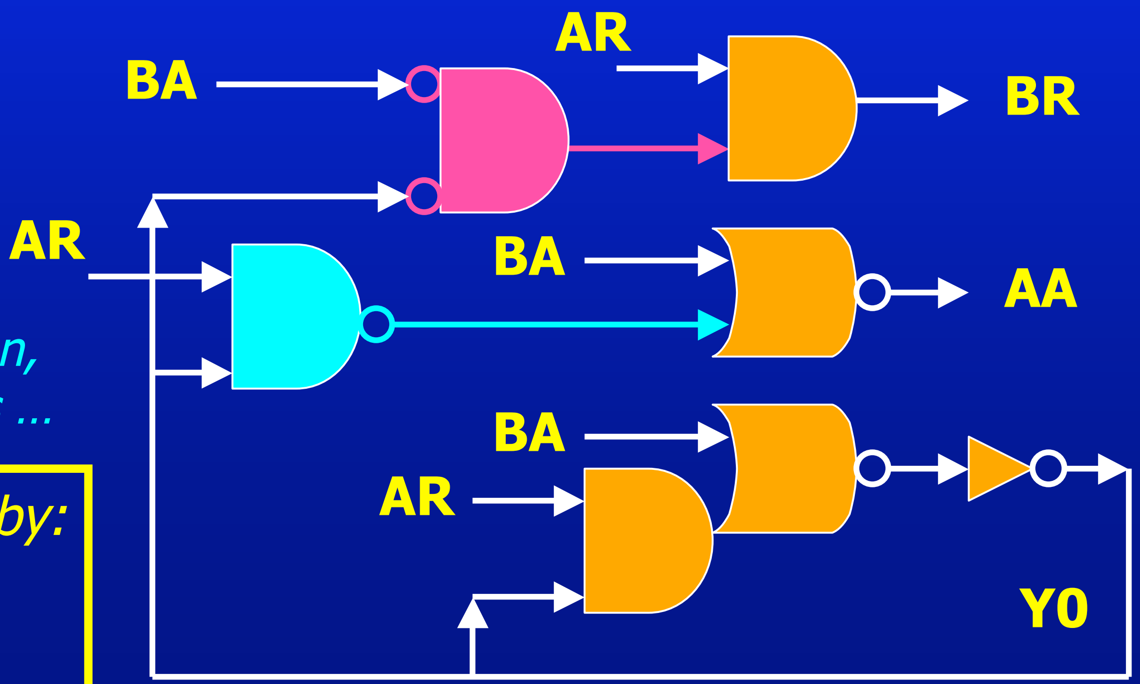
martin_q_element-L

Example #1: Martin Q-Element

Martin Implementation:
("QDI")



Burst-Mode Implementation:
(Fund. Mode)

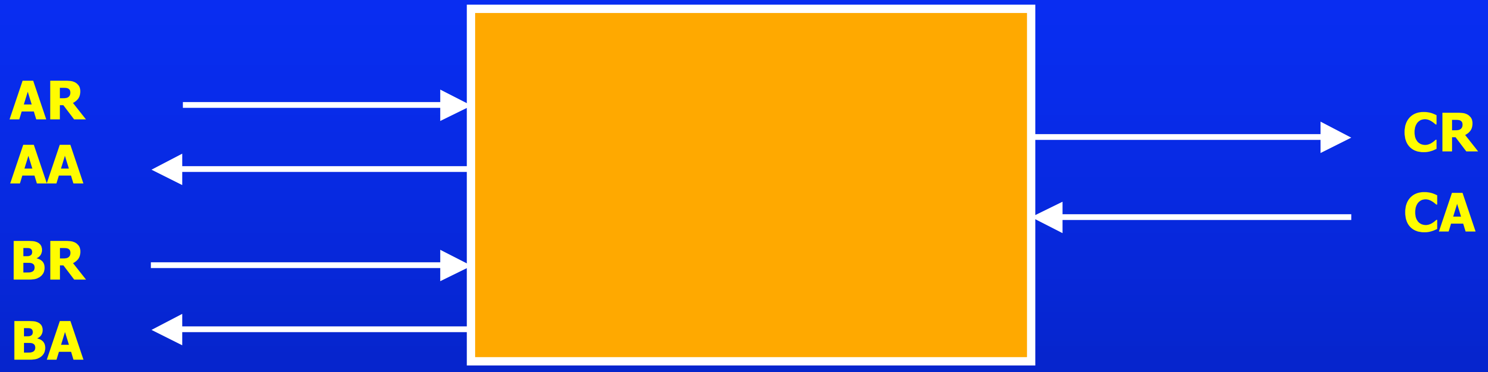


… after manual decomposition, to directly compare structures …

C-element *now replaced by:*
- **NOR2**
- **NAND2**

EXAMPLE #2A: Tangram Mixer

TANGRAM Mixer = "Call Element" (channel multiplexer)



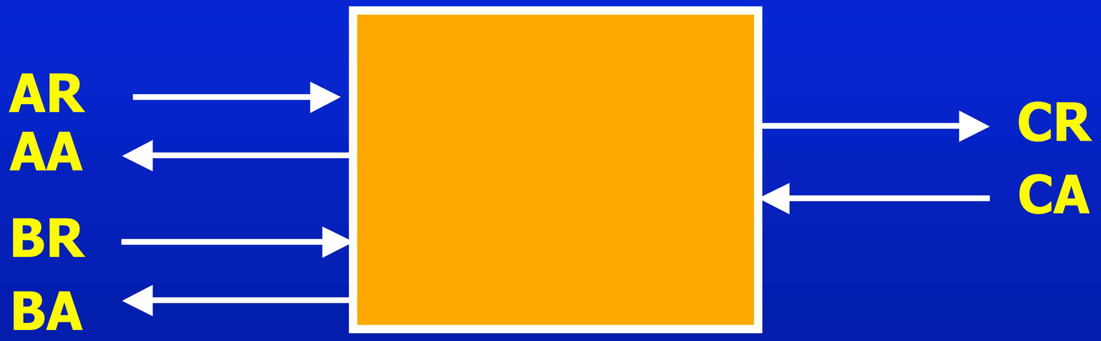
From Philips Research Lab: "TANGRAM" async tool flow ,

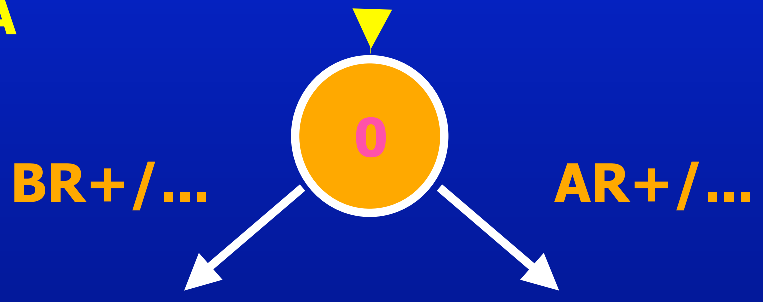
Kees van Berkel,
"VLSI Programming of Asynchronous Circuit for Low Power".
Chapter in "Asynchronous Digital Design," (eds. G. Birtwistle and A. Davis),
Springer-Verlag, Workshop in Computing Series,
pp. 152-210 (1995).

EXAMPLE #2A: Tangram Mixer

TANGRAM Mixer: *Deriving a BM Specification ...*

Protocol:
assumes at most one requesting channel (A or B) active at any time!

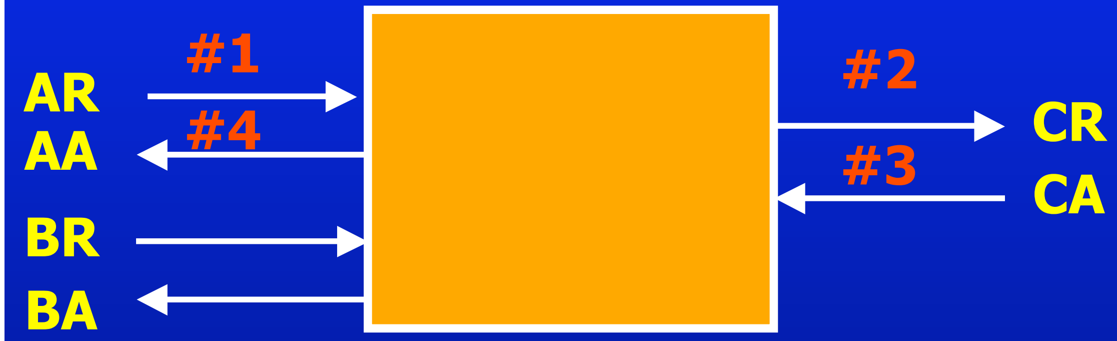


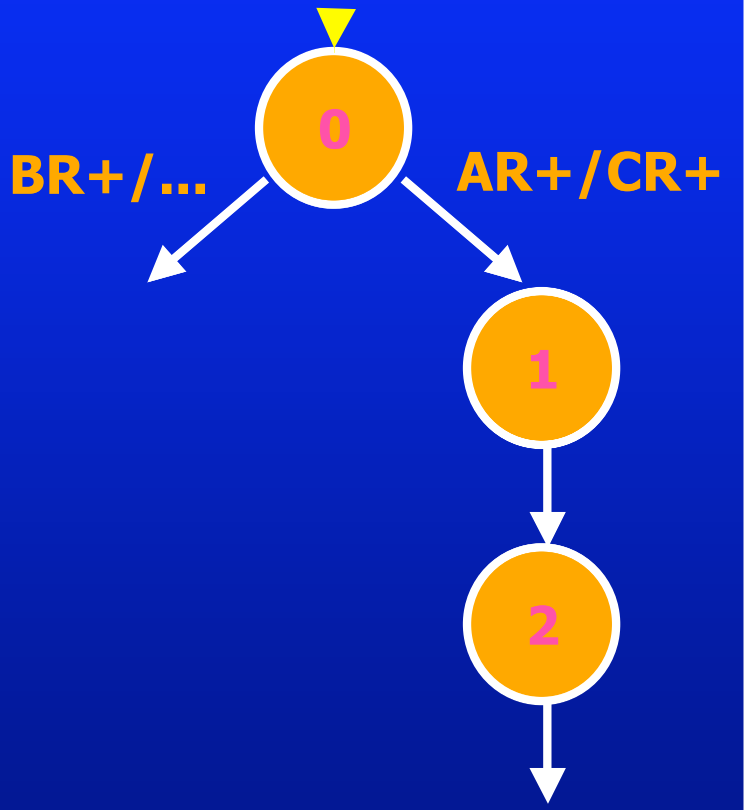


EXAMPLE #2A: Tangram Mixer

TANGRAM Mixer: *Deriving a BM Specification ...*

Protocol: for channel 'A' request





COMPLETE THE BURST-MODE SPEC ...
THEN CREATE A .bms FILE ...!

==> next slide

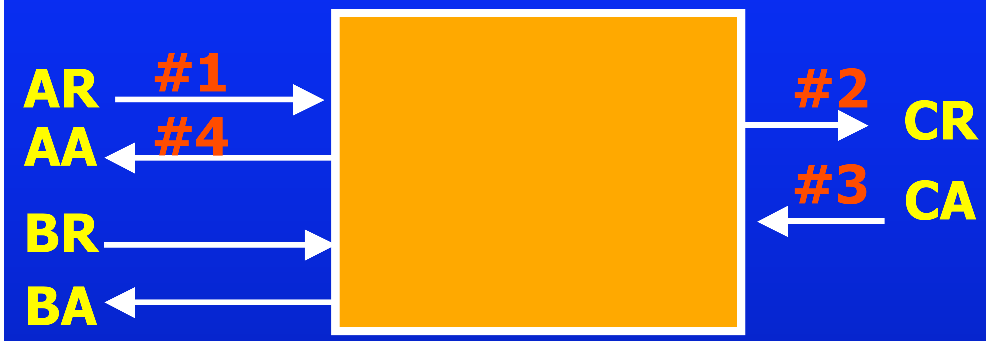
Example #2A: Tangram Mixer

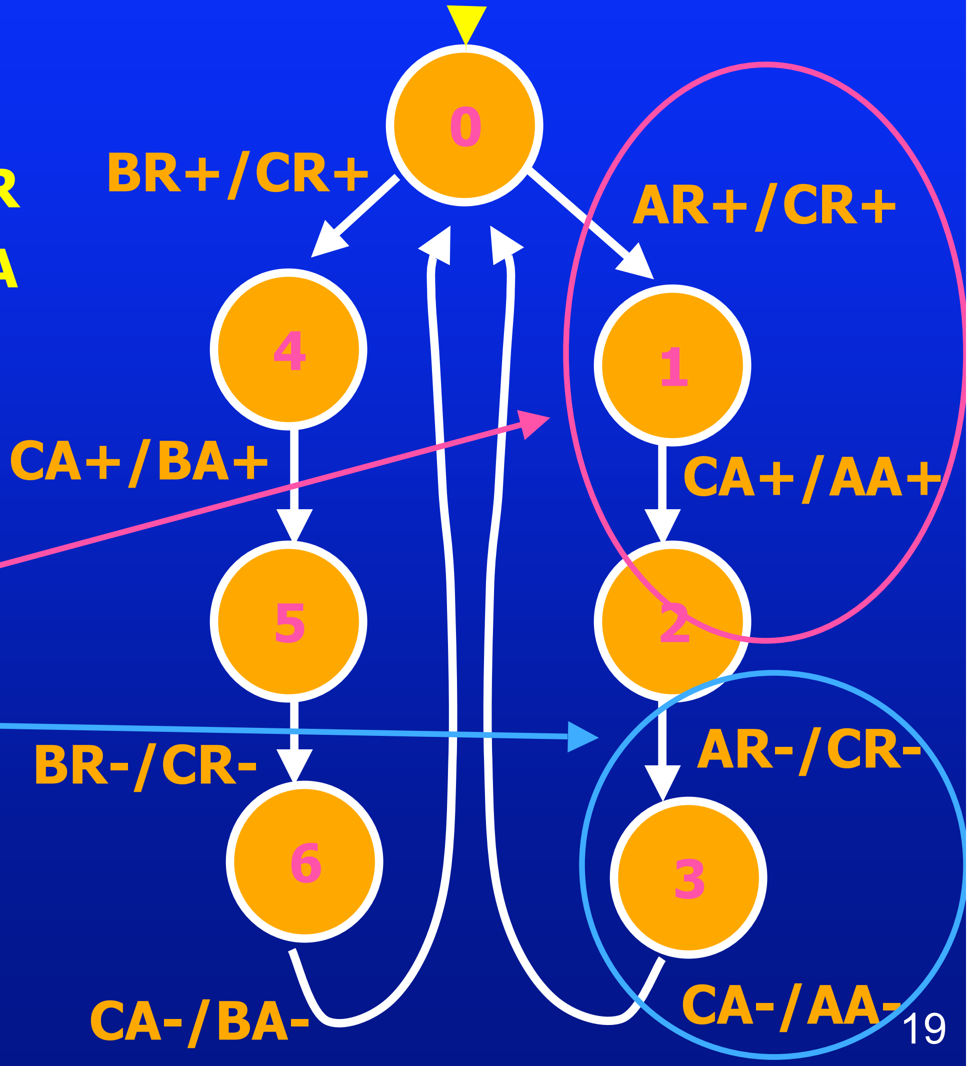
[... waiting while you create your BM spec;

when done, go to next slide.]

EXAMPLE #2A: Tangram Mixer

Deriving a Burst-Mode Specification…:





Channel A events #1-4…:

first, rising transitions

then, falling transitions

-use same protocol for both A and B requests

-assume at most one of A/B channels active at any time!

(see choice in state #0)

Example #2A: Tangram Mixer

Running MINIMALIST: the Simple Approach

Step #0. Getting Started ...

(a) go back into "min-demo" directory:

```
> cd ..
```

(b) create a new subdirectory:

```
> mkdir ex2A
```

(c) go to it:

```
> cd ex2A
```

(d) edit/create file for your BM spec:

```
> emacs (or vi) my-tangram-mixer.bms
```

When done: compare your own BMS spec with ...

../../examples/tangram-mixer/tangram-mixer.bms

Example #2A: Tangram Mixer

Step #1. Show BM Specification:

Graphic Display:

```
> bms2ps  my-tangram-mixer.bms
```

Step #2. Synthesize BM Implementation:

```
> minimalist-basic my-tangram-mixer.bms
```

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit: [follow instructions]

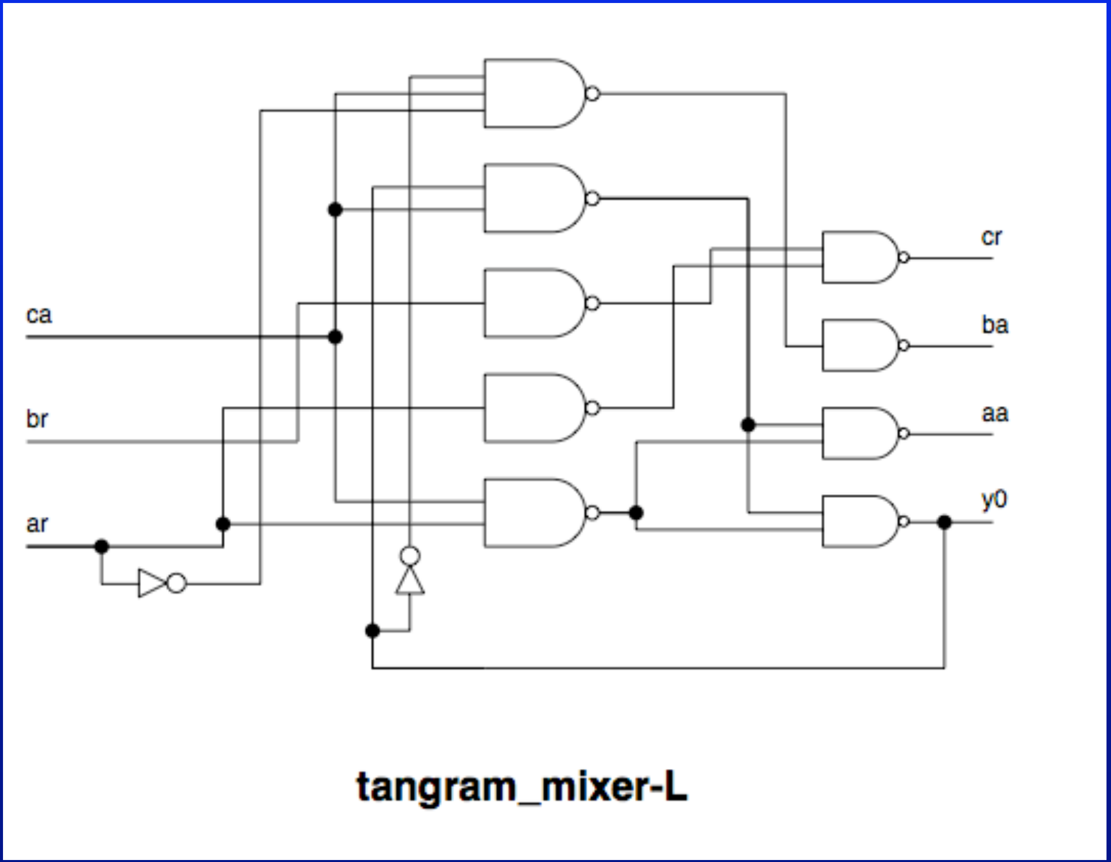
```
> plot_nand my_tangram_mixer-L.sol
```

Example #2A: Tangram Mixer

Running MINIMALIST: the Simple Approach

Step #3. Display It (cont.):

(b) Result of "plot_nand":



Example #2A: Tangram Mixer

Now, do another synthesis run, and compare:

use an *"area-oriented" script,* with fedback outputs

… first, get help on how to call the script:

```
> help minimalist-area
```

Step #2. Synthesize BM Implementation:

```
> minimalist-area my-tangram-mixer.bms  multi-output  fedback
```

Script is selected to run with:

- "multi-output" = shared logic
- "fedback" = try to use outputs as fedback state variables

Example #2A: Tangram Mixer

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

```
> [see displayed output]
```

NOTE: now no state variables are needed.

(b) Plot NAND/NAND Circuit: [follow instructions]

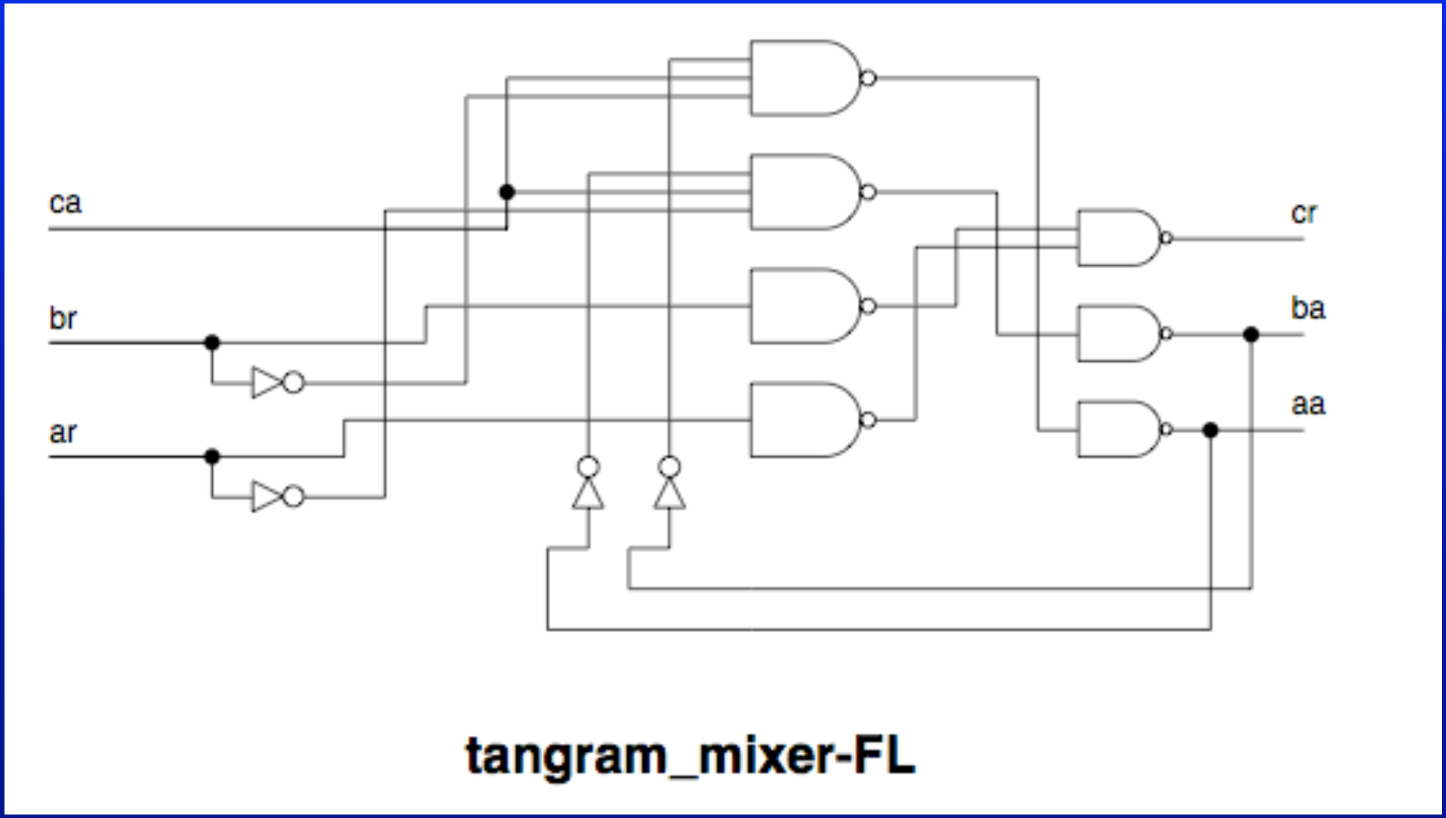
```
> plot_nand  my_tangram_mixer-FL.sol
```

Example #2A: Tangram Mixer

Running MINIMALIST: the Simple Approach

Step #3. Display It (cont.):

(b) Result of "plot_nand":

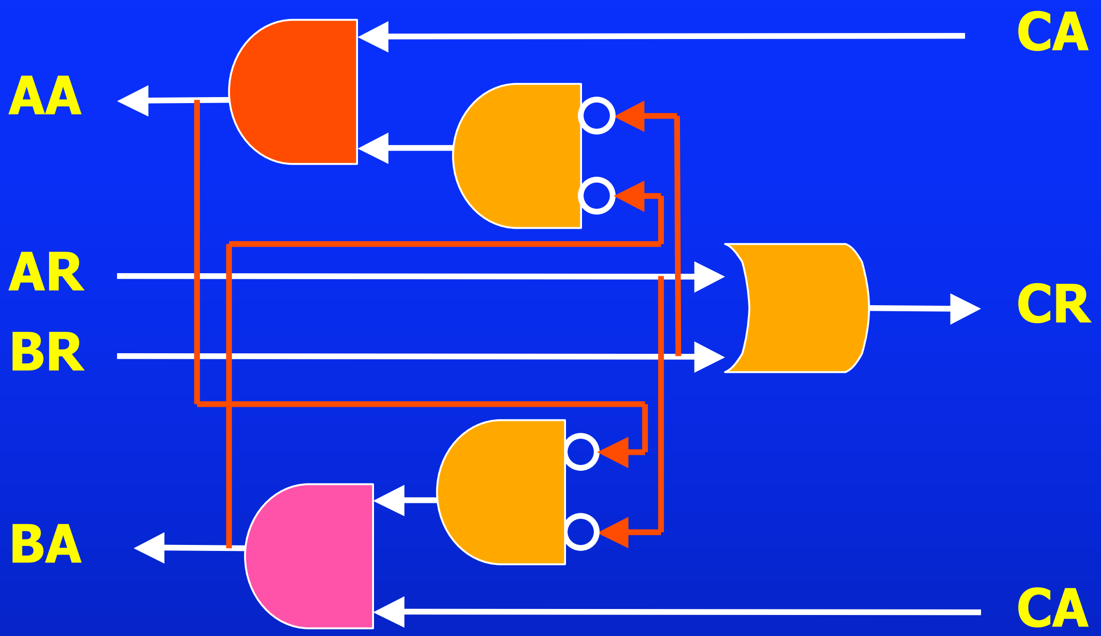


Example #2A: Tangram Mixer

MINIMALIST Implementation:

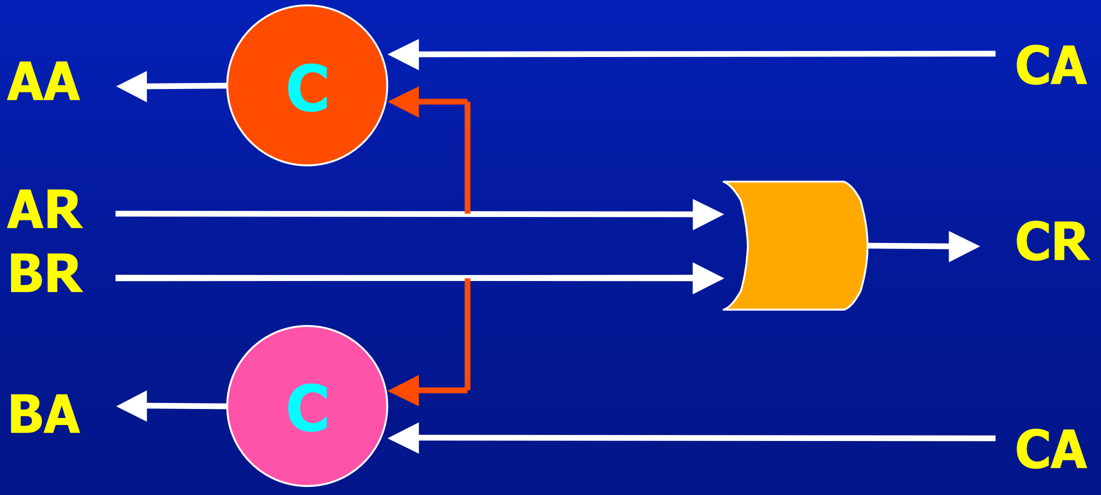
(Fund. Mode)

... after manual decomposition, to directly compare structures ...



Tangram Implementation:

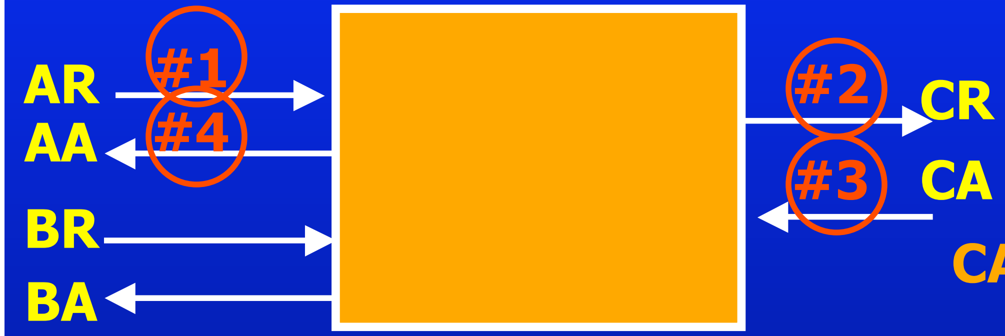
("QDI")

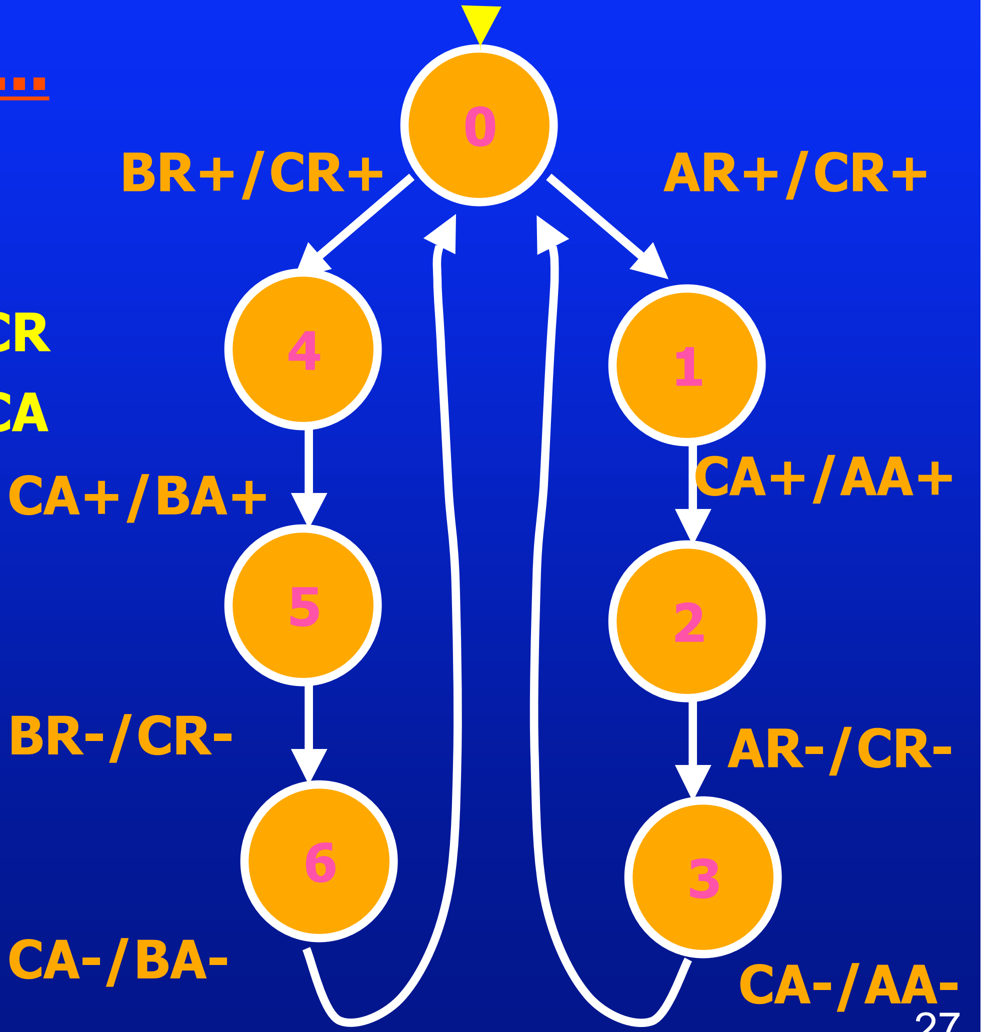


EXAMPLE #2B: Concurrent Mixer

Now... create a more concurrent BM Specification! ...

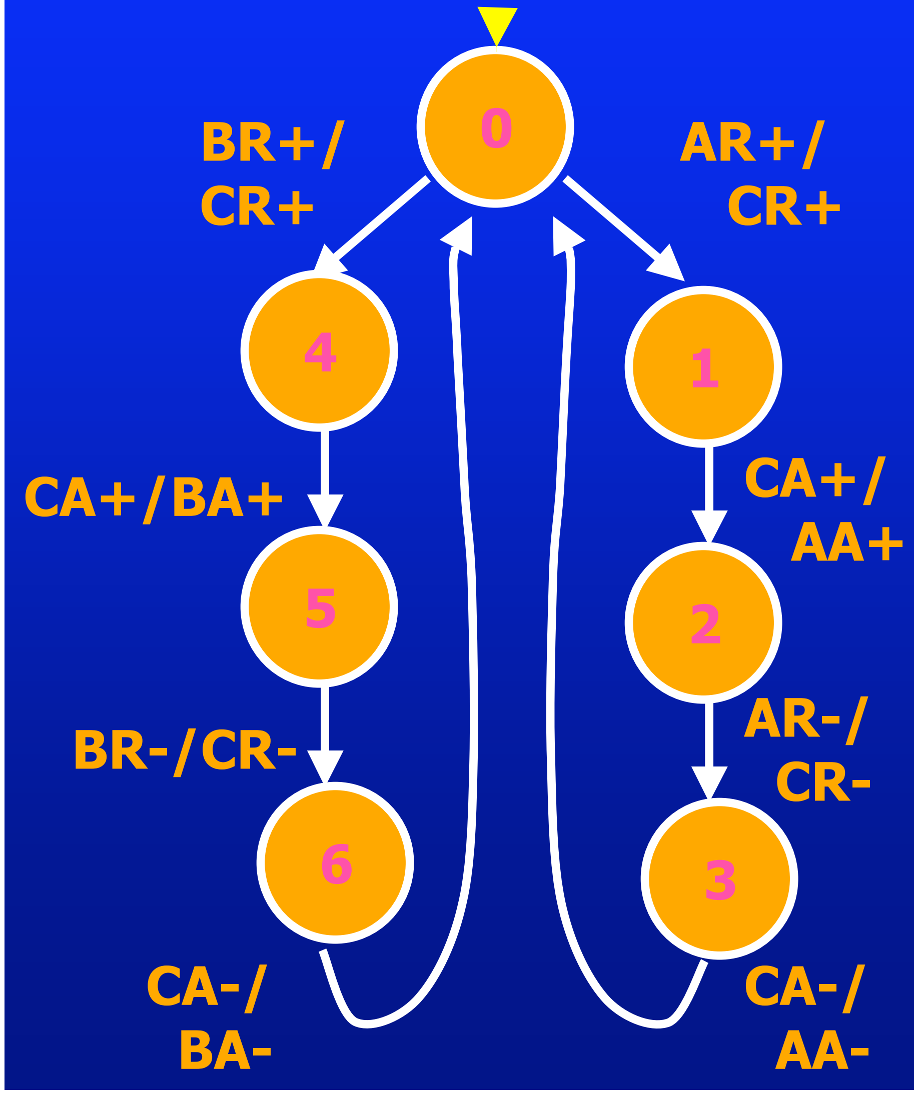
Basic Protocol: events #1-4...





EXAMPLE #2B: Concurrent Mixer

Basic Protocol:

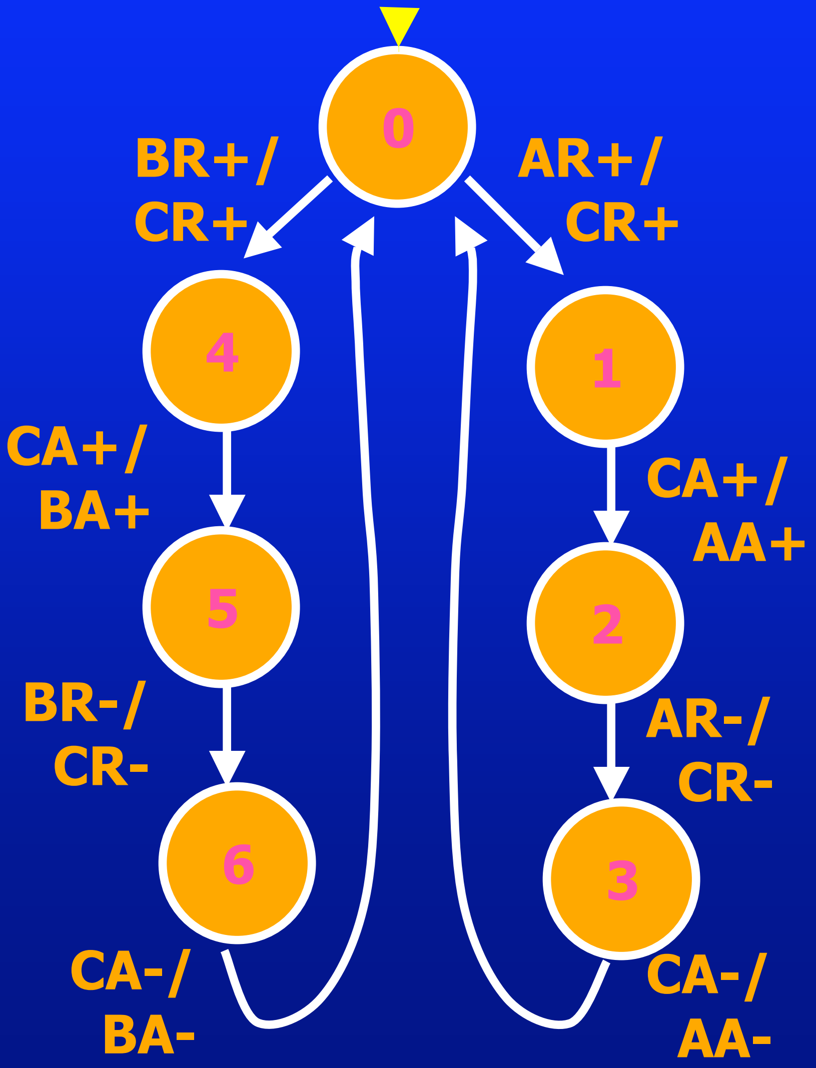


Concurrent Protocol:

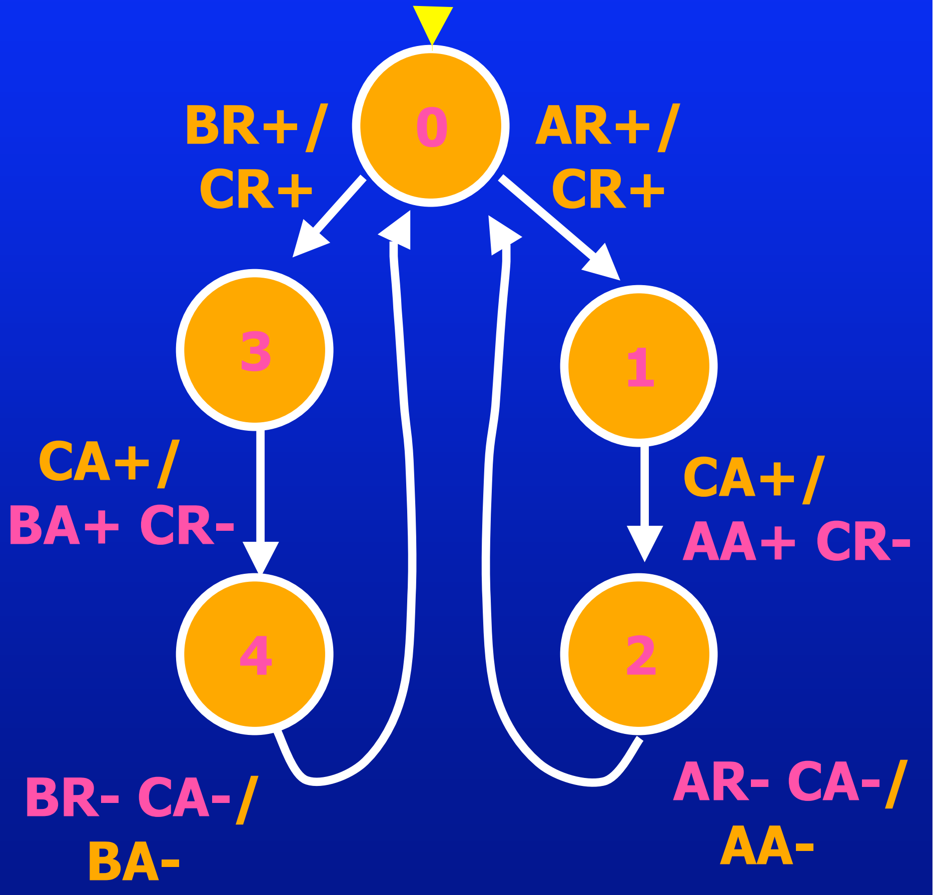
[…waiting while you create a more concurrent Burst-Mode specification for a mixer; when done, go to next slide.]

EXAMPLE #2B: Concurrent Mixer

Basic Protocol: BM Spec



Concurrent Protocol: BM Spec



EXAMPLE #2B: Concurrent Mixer

Running MINIMALIST: the Simple Approach

Step #0. Getting Started ...

(a) go back into "min-demo" directory:

```
> cd ..
```

(b) create a new subdirectory:

```
> mkdir ex2B
```

(c) go to it:

```
> cd ex2B
```

(d) edit/create file for your BM spec:

```
> emacs (or vi) my-concur-mixer.bms
```

When done: compare your own BMS spec with ...

../../examples/concur-mixer/my-concur-mixer.bms

Example #2B: Concurrent Mixer

Step #1. Show BM Specification

(a) Graphic Display:

```
> bms2ps  my-concur-mixer.bms
```

Step #2. Synthesize BM Implementation:

```
> minimalist-basic  my-concur-mixer.bms
```

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit: [follow instructions]

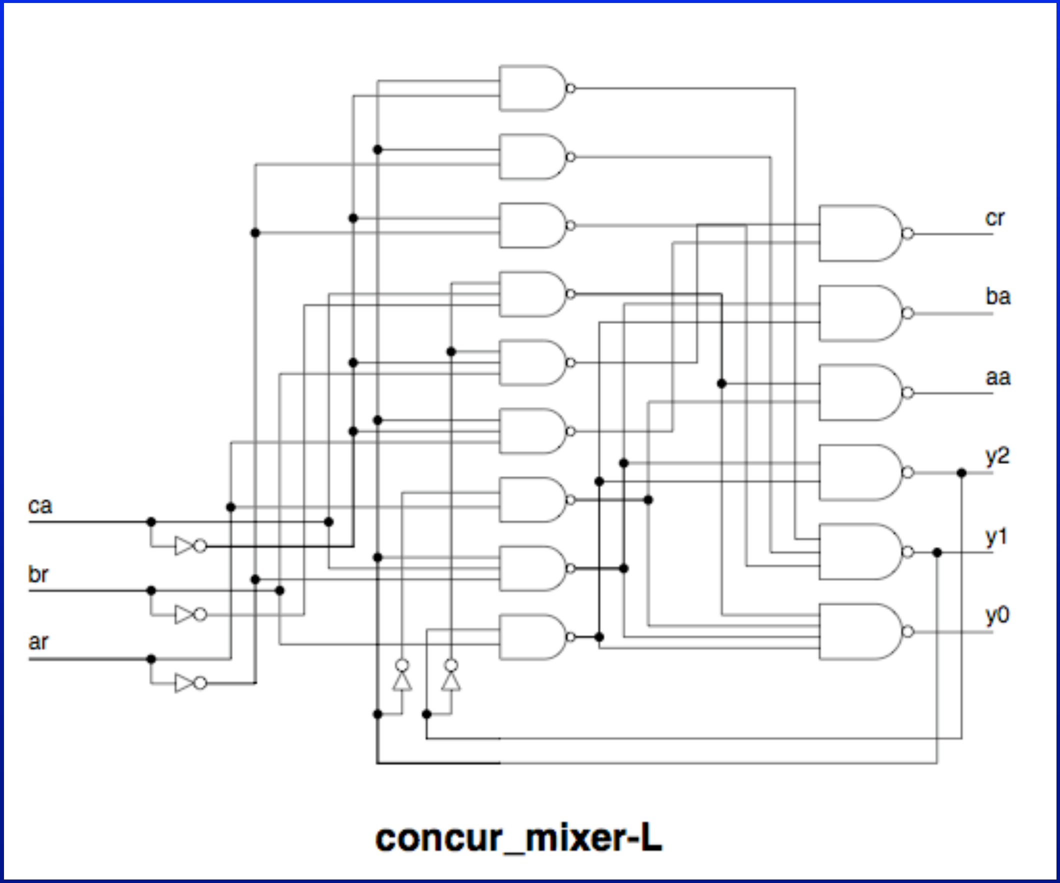
```
> plot_nand my_concur_mixer-L.sol
```

EXAMPLE #2B: Concurrent Mixer

Running MINIMALIST: the Simple Approach

Step #3. Display It (cont.):

(b) Result of "plot_nand":



Example #2B: Concurrent Mixer

Now, do <u>another</u> synthesis run…:

using an *"area-oriented" script* (and compare results)….

<u>Step #2. Synthesize BM Implementation:</u>

```
> minimalist-area my-concur-mixer.bms  multi-output  fedback
```

<u>Step #3. Display It:</u>

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit: [follow instructions]

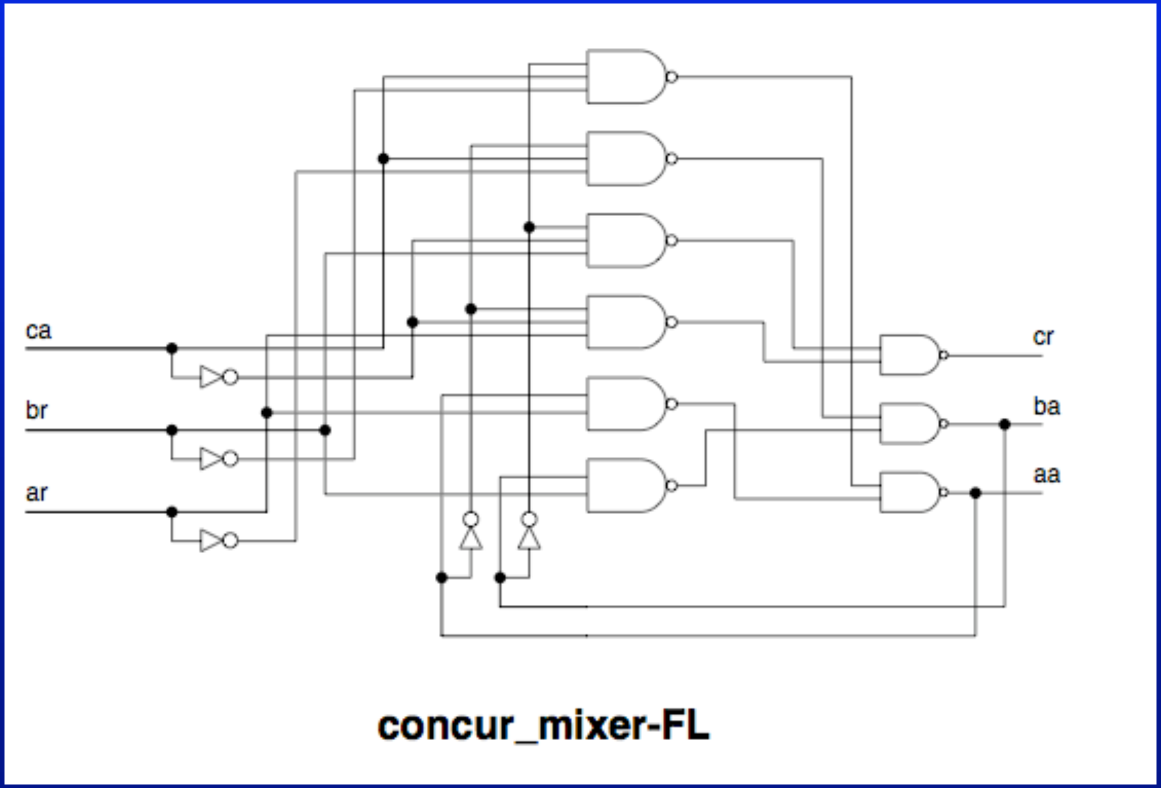
```
> plot_nand  my_concur_mixer-FL.sol
```

EXAMPLE #2B: Concurrent Mixer

Running MINIMALIST: the Simple Approach

Step #3. Display It (cont.):

(b) Result of "plot_nand":

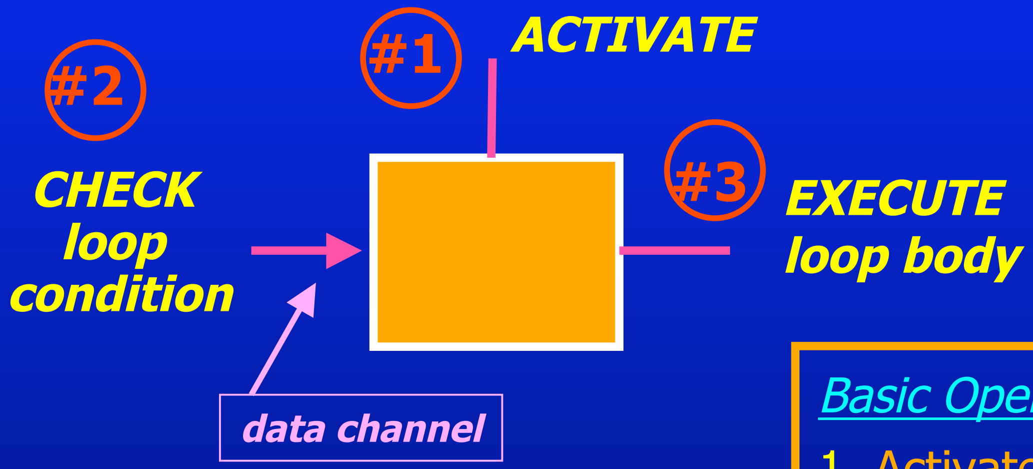


EXAMPLE #2C: "While" Module

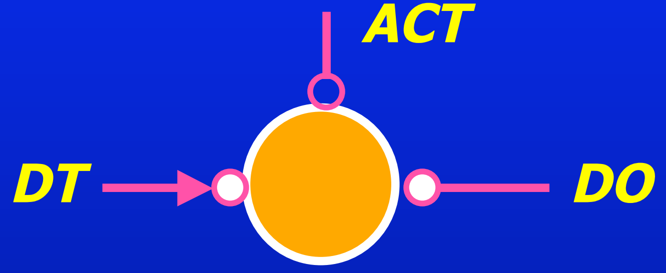
Now, you will design an entire BM specification from start-to-finish … !

Problem: Design a Burst-Mode "While" (i.e., Loop) Controller

Simple Block Diagram *(with channels):*



Balsa/Tangram Equivalent:



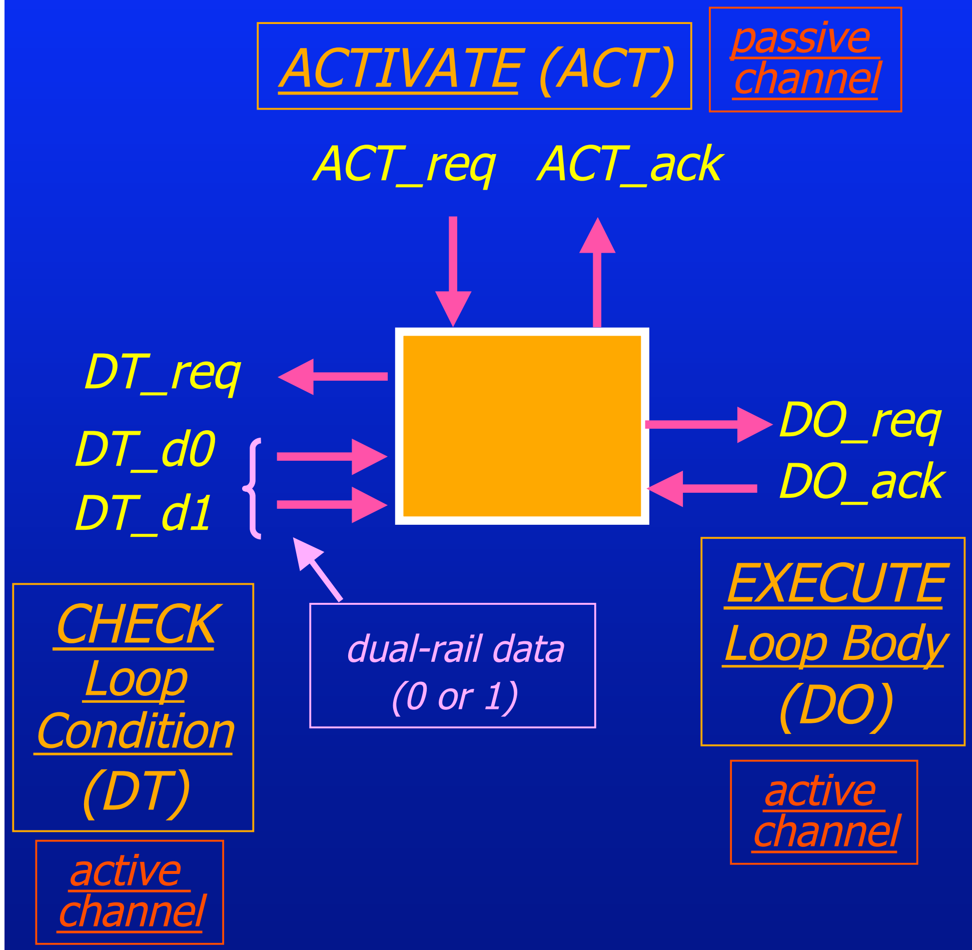
Basic Operation:

1. Activate "while" component
2. Check loop variable
   - if 0, exit (go to #1)
   - if 1, continue to #3
3. Execute loop body

Repeat #2/#3 until loop var false

EXAMPLE #2C: "While"Module

Detailed Block Diagram:



DETAILED OPERATION:

1. Wait until module activated (ACT_req+)
2. Request loop variable (DT_req+)

   a. **if loop variable = 0** (DT_d0+)
      - complete handshake on DT
      - complete handshake on ACT
      - return to #1 (wait for next activation)

   b. **if loop variable = 1** (DT_d1+)
      - complete handshake on DT
      - *execute loop body* (i.e., do full handshake on DO)
      - go to #2 (start next loop test)

EXAMPLE #2C: "While"Module

OPTIMIZATIONS (optional):

A. You may overlap these 2 handshakes (i.e. make them concurrent)

B. You may overlap these 2 handshakes (i.e. make them concurrent), as long as:
- **DT** is reset (and not changing) during the active phase of **DO**

DETAILED OPERATION:

1. Wait until module activated (ACT_req+)
2. Request loop variable (DT_req+)

   a. if loop variable = 0 (DT_d0+)
   - complete handshake on DT
   - complete handshake on ACT
   - return to #1 (wait for next activation)

   b. if loop variable = 1 (DT_d1+)
   - complete handshake on DT
   - *execute loop body* (i.e., do full handshake on DO)
   - go to #2 (start next loop test)

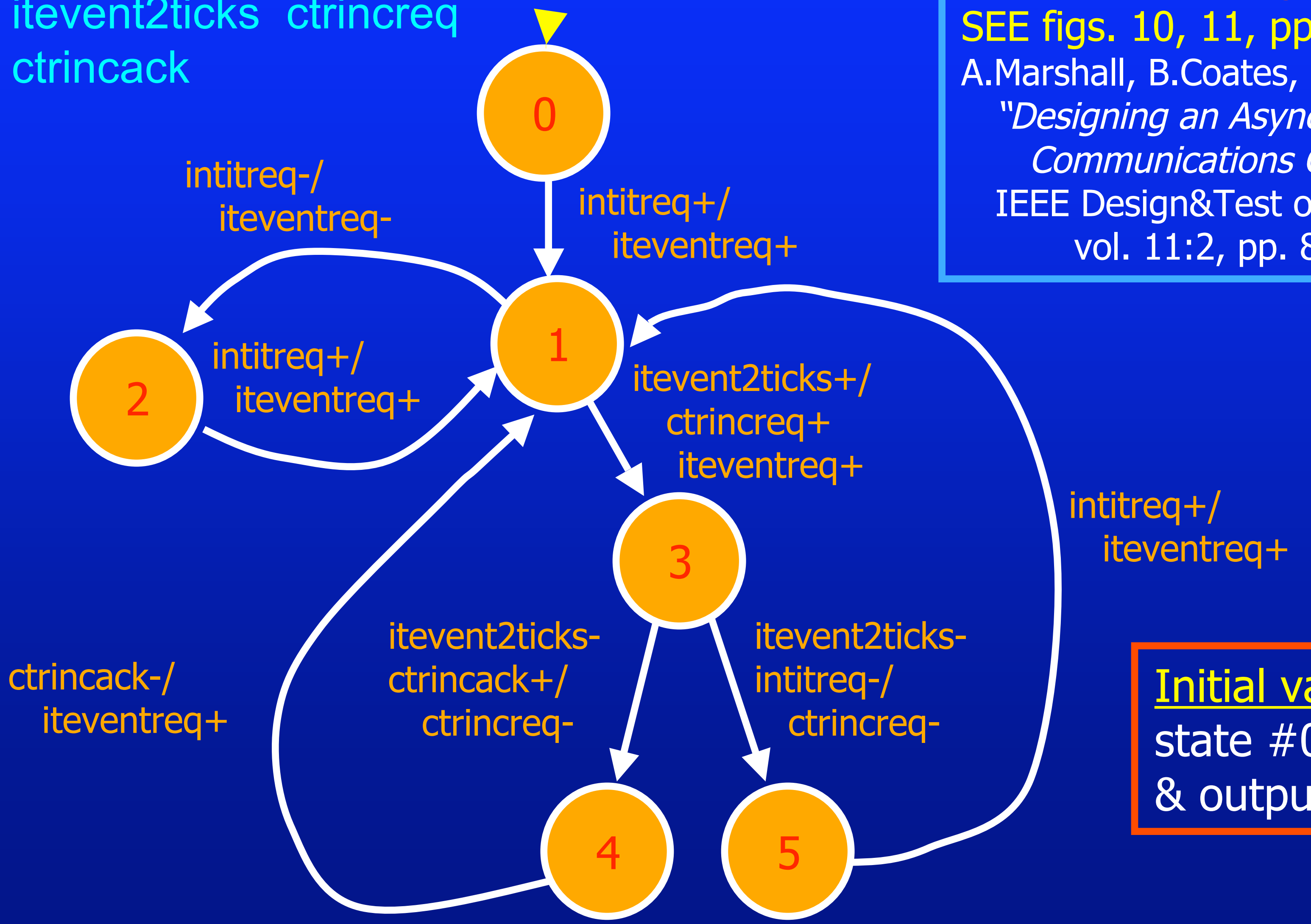
Hint: for B., you may need to unroll the loop once to apply optimization

Example #3: "HP-IR" (HP Labs)

Inputs:	Outputs:
intitreq	iteventreq
itevent2ticks	ctrincreq
ctrincack	

From HP Labs/Stanford "Stetson" Project:

SEE figs. 10, 11, pp. 17-18:
A.Marshall, B.Coates, P.Siegel, *"Designing an Asynchronous Communications Chip"*, IEEE Design&Test of Computers, vol. 11:2, pp. 8-21 (1994)



Initial values: in state #0, all inputs & outputs are 0

Example #3: HP-IR

Running MINIMALIST: the Simple Approach

Step #0. Getting Started ...

(a) go back into "min-demo" directory:

```
> cd ..
```

(b) create a new subdirectory:

```
> mkdir ex3
```

(c) go to it:

```
> cd ex3
```

(d) copy the BM spec:

```
> cp ../../examples/additional-benchmark-exs/hp-ir/hp-ir.bms  .
```

Step #1. Show BM Specification

(a) Graphic Display:

```
> bms2ps  hp-ir.bms
```

Example #3: HP-IR

Running MINIMALIST: the Simple Approach

Several Useful Scripts:

Produce 1 -- or 4 -- Circuit Implementations For Each Script

- fedback vs. non-fedback outputs
- single-output vs. ‘output-disjoint’ vs. ‘multi-output’ (shared products)
- different cost functions, ...

(a) BASIC (critical race-free): “minimalist-crf”,“minimalist-crf-suite”

> no optimal state assignment

(b) SPEED: “minimalist-speed”, “minimalist-speed-suite”

(c) AREA: “minimalist-area”, “minimalist-area-suite”

Example #3: HP-IR

Running MINIMALIST: the Simple Approach

Step #2. Synthesize BM Implementation: *[try each!]*

- Try "help <scriptname>" for details + parameters:

```
>minimalist-crf
>minimalist-crf-suite
>minimalist-speed
>minimalist-speed-suite
>minimalist-area
>minimalist-area-suite
```

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit:

> [follow instructions]

Example #3: HP-IR

Running MINIMALIST: the Simple Approach

Step #4. Convert to Verilog Output: *[optional]*

```
> pla2verilog <.sol file name>
```

OR:

Step #4. Add Initialization Circuitry + Convert to Verilog Output: *[optional]*

```
> pla2verilog -Init2  <.sol file name>  -R hp-ir.bms
```

GETTING HELP:

…extensive *on-line help* available at all times; type:

```
> help
```

for a list of options. Type >help <cmd-name> for detailed help on a command.

Example #3: HP-IR

Running MINIMALIST: the Simple Approach

Step #5. Verify circuit against spec: *[optional]*

```
> bms-verify  hp-ir.bms  <.sol file name>
```

Example #3: HP-IR

"Generalized C-Element" Implementations:

Produces "SET"/"RESET" functions for each output + next-state

- see part 1 of the tutorial for details…

2 Common Implementation Styles:

(i) gC-element: "SET"=pull-down, "RESET"=pull-up

- gC output: has *inverter* + *"keeper"* (to hold state)

(ii) using C-element (2-input):

- implement "SET" and "RESET" as separate logic networks
- each network is 2-level, feeds into 2-input C-element
- "RESET": must invert, before feeding into C-element

(d) GC SCRIPTS: "minimalist-gc", "minimalist-gc-suite"

Example #3: HP-IR

Running MINIMALIST: the Simple Approach

Step #2. Synthesize BM Implementation: *[try each!]*

- Try "help <scriptname>" for details + parameters:

```
> minimalist-gc
> minimalist-gc-suite
```

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit:

> [follow instructions]

NOTE: (b) currently only displays separate SET and RESET functions (not GC elements!)

Example #4: RF-Control

Running MINIMALIST:

Custom Synthesis Using the MINIMALIST "Shell"

From HP Labs/Stanford

"Stetson" Project

A.Marshall, B.Coates, P.Siegel, *"Designing an Asynchronous Communications Chip"*, IEEE Design&Test of Computers, vol. 11:2, pp. 8-21 (1994)

Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #0. Getting Started ...

(a) go back into "min-demo" directory:

```
> cd ..
```

(b) create a new subdirectory:

```
> mkdir ex4
```

(c) go to it:

```
> cd ex4
```

(d) copy the BM spec:

```
> cp ../../examples/additional-benchmark-exs/hp-ir/rf-control.bms  .
```

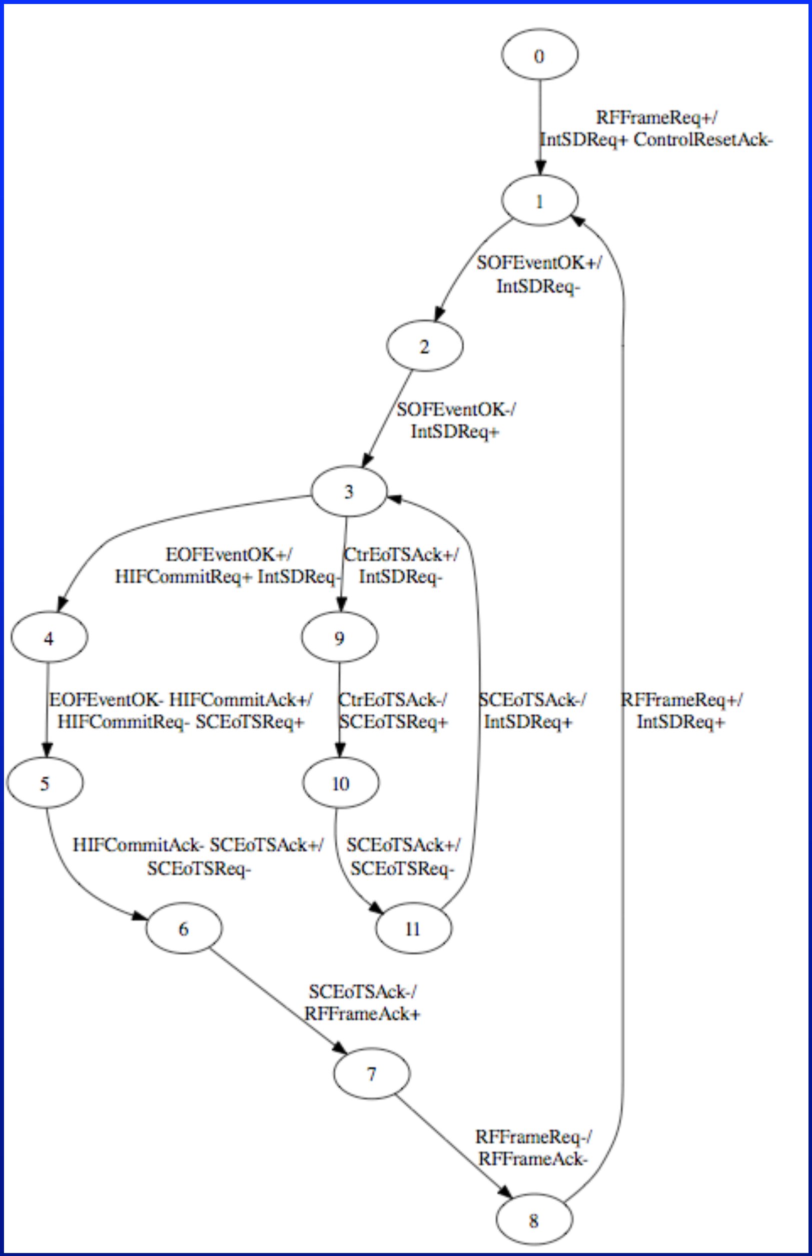
Step #1. Show BM Specification

(a) Graphic Display:

```
> bms2ps  rf-control.bms
```

Example #4: RF-Control

Step #1. Show BM Specification



Example #4: RF-Control

Step #1. Show BM Specification

(a) Look at "BMS" text file:

```
> more  rf-control.bms
```

```
name RF_control
Input     RFFrameReq       0
Input     SOFEventOK       0
Input     EOFEventOK       0
Input     CtrEoTSAck       0
Input     SCEotSAck        0
Input     HIFCommitAck     0

Output    ControlResetAck  1
Output    RFFrameAck       0
Output    IntSDReq         0
Output    SCEoTSReq        0
Output    HIFCommitReq     0
```

[… continued on right column ==>]

```
0  1     RFFrameReq+ | IntSDReq+ ControlResetAck-
1  2     SOFEventOK+ | IntSDReq-
2  3     SOFEventOK-  | IntSDReq+
3  4     EOFEventOK+ | HIFCommitReq+ IntSDReq-
4  5     EOFEventOK- HIFCommitAck+ |
                 HIFCommitReq+   SCEoTSReq+
5  6     HIFCommitAck- SCEoTSAck+ | SCEoTSReq+
6  7     SCEoTSAck- | RFFrameAck+
7  8     RFFrameReq- | RFFrameAck-
8  1     RFFrameReq+ | IntSDReq+
3  9     CtrEoTSAck+ | IntSDReq-
9  10    CtrEoTSAck- | SCEoTSReq+
10 11    SCEoTSAck+ | SCEoTSReq-
11 3     SCEoTSAck- | IntSDReq+
```

Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #2. Customize Synthesis of BM Implementation: basic run

(… still in "MinShell" …: *type "help" for each command below*)

```
> read-spec  rf-control.bms
> help …
  ….
> min-states          #basic state minimization
> help assign-states
> assign-states       #basic (crit race-free) state assignment (not optimal)
> help min-logic
> min-logic           #basic logic minimization (using default parameters)
```

Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit:

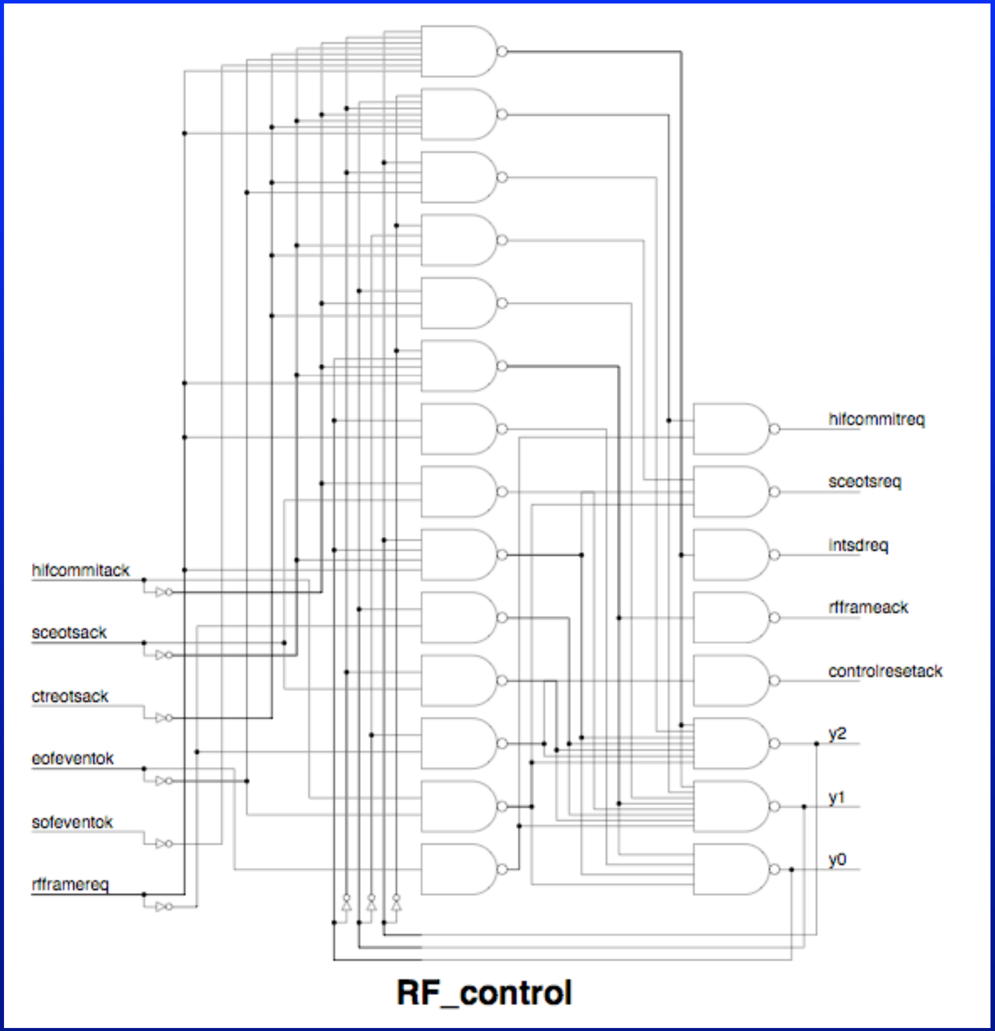
> [follow instructions]

Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #3. Display It (cont.):

(b) Result of "plot_nand":



Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #2'. A Different Customized Run: targeting speed

- use optimal state assignment (targeted to primary outputs)
- no fedback outputs
- no logic sharing (-s)
- target critical I-to-O paths (-P)

```
 > read-spec  rf-control.bms
> help …
> min-states                       #state minimization
> help assign-states
> assign-states -O -P -S -s #optimal state assignment (-O)
                                    #target output logic only (-S)
                                    #single-output logic (-s)
                                    #target critical I/O paths (-P)
> help min-logic
> min-logic -P -s                 # better logic minimization:
                                    #single-output logic (-s)
                                   #target critical I/O paths (-P)
```

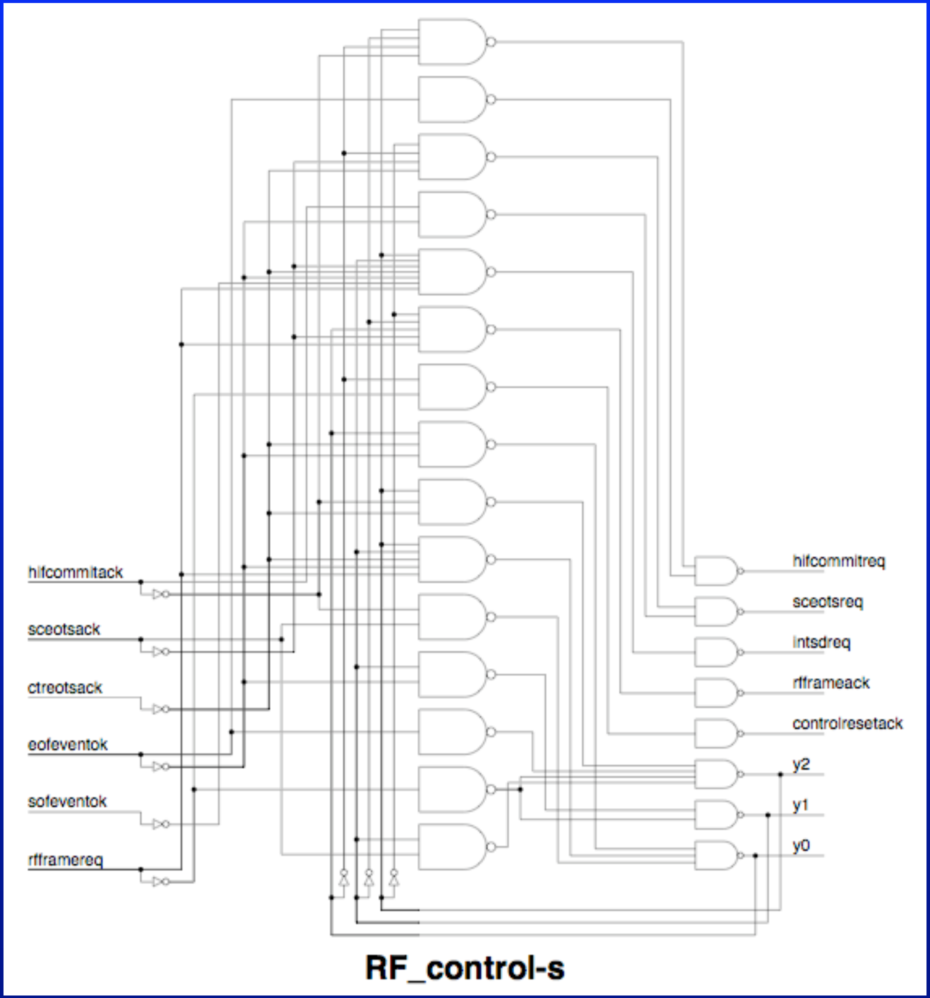
Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #3'. Display It (cont.):

(b) Result of "plot_nand":

Obtains faster logic
for the primary outputs…



Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #2". Yet Another Custom Run: targeting area

- use optimal state assignment (targeted to both outputs and next-state)
- fedback outputs (to eliminate some explicit state variables)
- logic sharing = multi-output (no -d or -s)
- target total # of literals (-L)

```
> read-spec  rf-control.bms
> help ...
> min-states -F                 #state minimization, w/fedback
                                          #outputs as state vars (-F)
> help assign-states
> assign-states -F -L -O       #optimal state assignment (-O)
                                         #w/fedback outputs (-F)
                                         #shared products (no -d or -s)
                                         # (i.e. "multi-output")
                                         #target total literal count (-L)
> help min-logic
> min-logic -F -L                    #w/fedback outputs (-F)
                                         #shared products (no -d or -s)
                                         #target total literal count (-L)
```

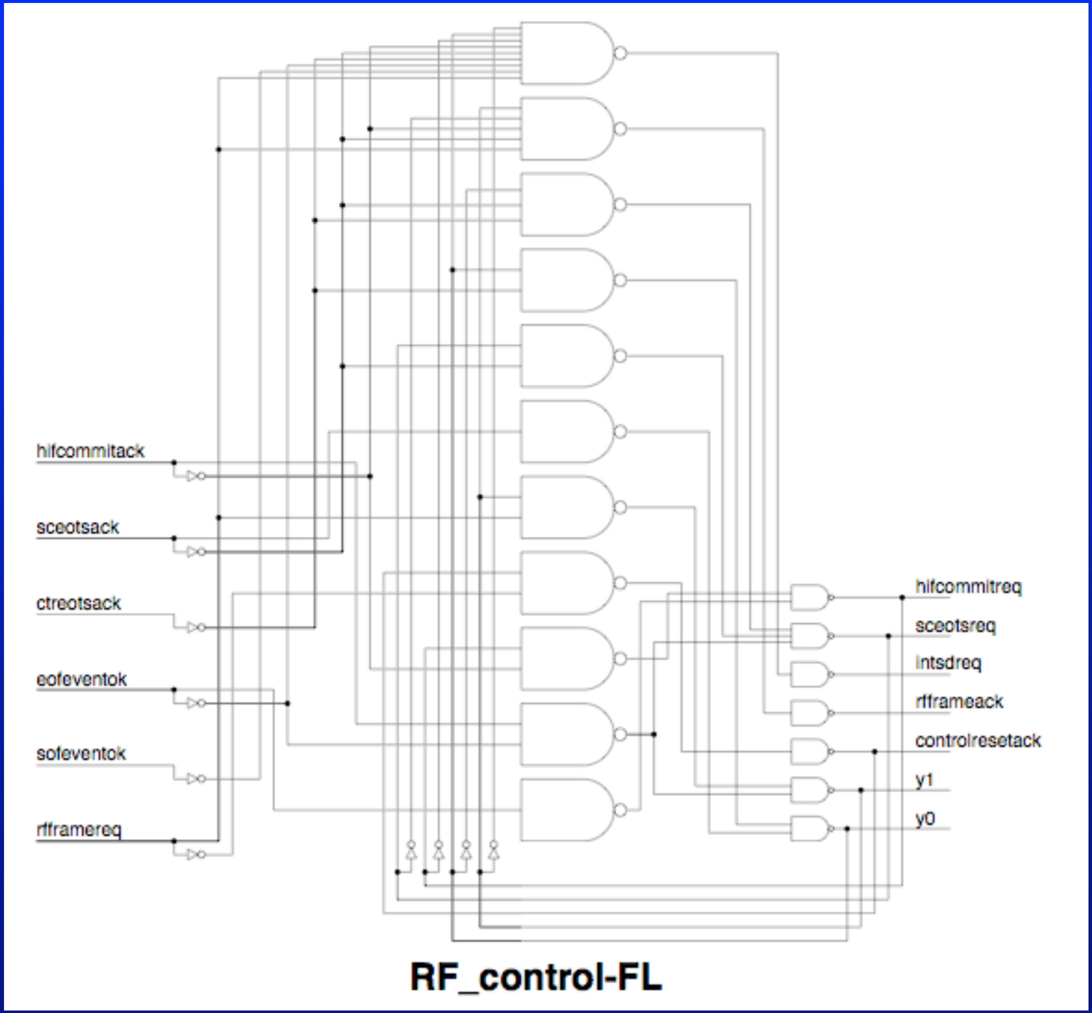
Example #4: RF-Control

Running MINIMALIST: Using the MINIMALIST "Shell"

Step #3'. Display It (cont.):

(b) Result of "plot_nand":

Obtains smaller total area…



Example #5: Handling Large Examples ("P1 Controller")

From HP Labs/Stanford "Stetson"Project

A.Marshall, B.Coates, P.Siegel,
"Designing an Asynchronous Communications Chip",
IEEE Design&Test of Computers, vol. 11:2, pp. 8-21 (1994)

Step #0. Getting Started ...

(a) go back into "min-demo" directory:

```
> cd ..
```

(b) create a new subdirectory:

```
> mkdir ex3
```

(c) go to it:

```
> cd ex3
```

(d) copy the BM spec:

```
> cp ../../examples/additional-benchmark-exs/stetson/p1.bms  .
```

Example #5: Handling Large Examples ("P1 Controller")

Step #1. Show BM Specification

(a) Look at "BMS" text file:

```
> more  p1.bms
```

Summary: a large async controller

13 inputs

14 outputs

33 (burst-mode) states

Step #1. Show BM Specification (cont.)

(b) Graphic Display:

```
> bms2ps  p1.bms
```

Example #5: Handling Large Examples ("P1 Controller")

Synthesis Strategies for Faster Runtime

(i) *avoid* using "optimal" state assignment

(... instead, just use simple "non-optimal" critical race-free assignment)

(ii) try to avoid multi-output shared logic (expensive to run)

(... instead, use "single-output" [-s] or "output-disjoint [-d]", which are used in speed-oriented options.)

(iii) possibly, try using fedback outputs:

(... potential benefit: fewer states;
potential drawback (*sometimes*): more total inputs to controller ...)

Example #5: Handling Large Examples ("P1 Controller")

Initial Synthesis Run: Using SCRIPTS

Step #2. Synthesize BM Implementation:

```
>minimalist-crf  p1.bms  speed  fedback
```

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

(b) Plot NAND/NAND Circuit:

> [follow instructions]

Example #5: Handling Large Examples ("P1 Controller")

<u>An Alternative Run: Using the MINIMALIST "Shell"</u>

<u>Step #2. Synthesize BM Implementation:</u>

… one of many possible custom runs -- can try others too:

```
> read-spec  p1.bms

> help …
    ….
> min-states  -F          # state minimization: with fedback outputs
> help assign-states
> assign-states -F -C    #non-optimal state assignment (-C);
                           # assumes fedback outputs (-F)
> help min-logic
> min-logic -F -L -d      #logic minimization:  for literal optzn (-L);
                            # try output-disjoint logic (-d) [partial sharing];
                            # assumes fedback outputs (-F)
```

Example #5: Handling Large Examples (“P1 Controller”)

An Alternative Run: Using the MINIMALIST “Shell”

Step #3. Display It:

(a) 2-Level Equations + Results Summary:

> [see displayed output]

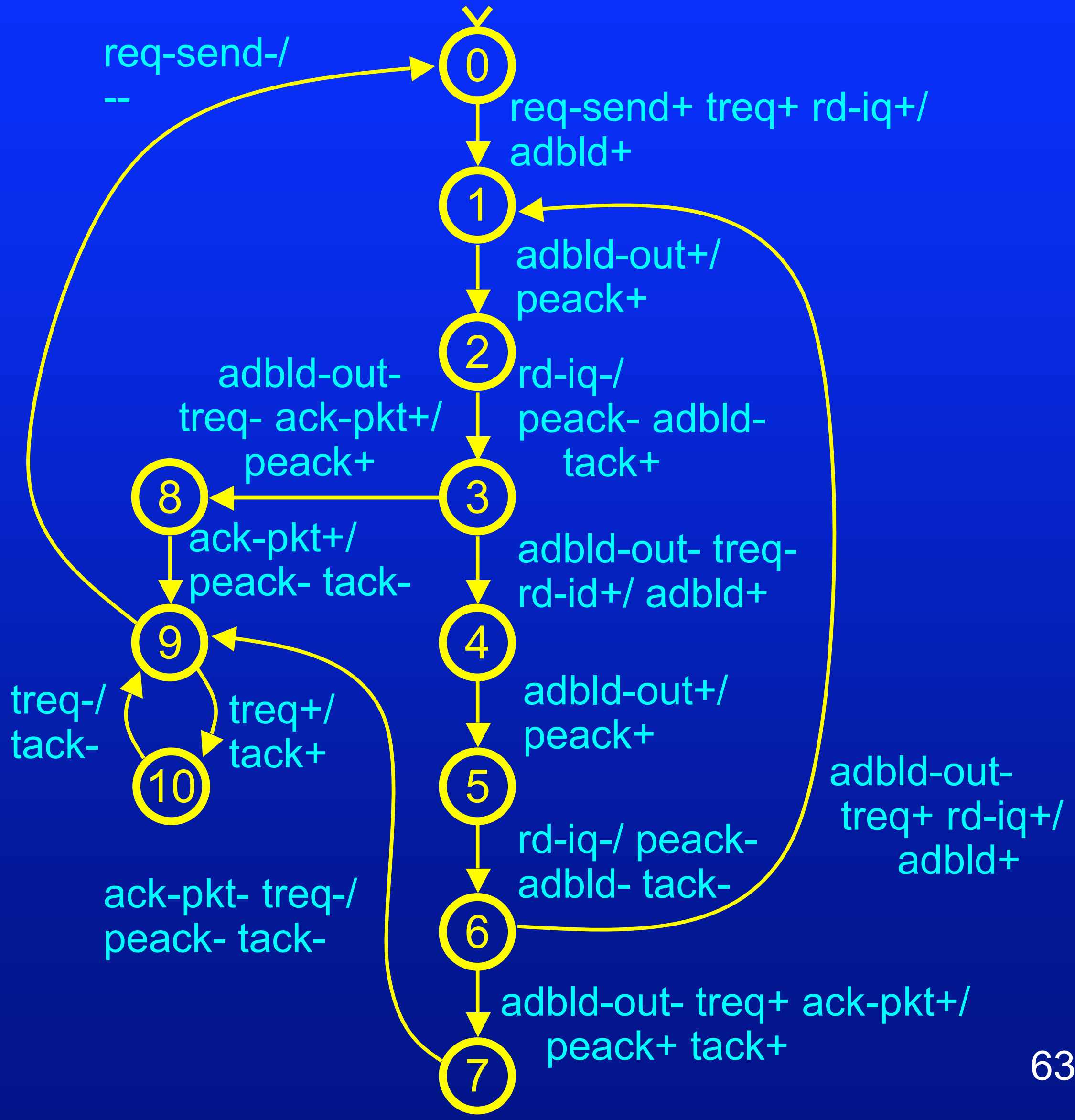
(c) Plot NAND/NAND Circuit:

> [follow instructions]

... Compare the 2 different implementations:
see Step #3(a) for statistics

Example #6: "PE-SEND-IFC" (HP Labs)

Inputs:	Outputs:
req-send	tack
treq	peack
rd-iq	adbld
adbld-out	
ack-pkt	



0
1
2
3
4
5
6
7
8
9
10
req-send-/ --
req-send+ treq+ rd-iq+/ adbld+
adbld-out+/ peack+
rd-iq-/ peack- adbld- tack+
adbld-out- treq- ack-pkt+/ peack+
ack-pkt+/ peack- tack-
adbld-out- treq- rd-id+/ adbld+
adbld-out+/ peack+
rd-iq-/ peack- adbld- tack-
adbld-out- treq+ rd-iq+/ adbld+
adbld-out- treq+ ack-pkt+/ peack+ tack+
treq-/ tack-
treq+/ tack+
ack-pkt- treq-/ peack- tack-

From HP Labs "Mayfly" Project:

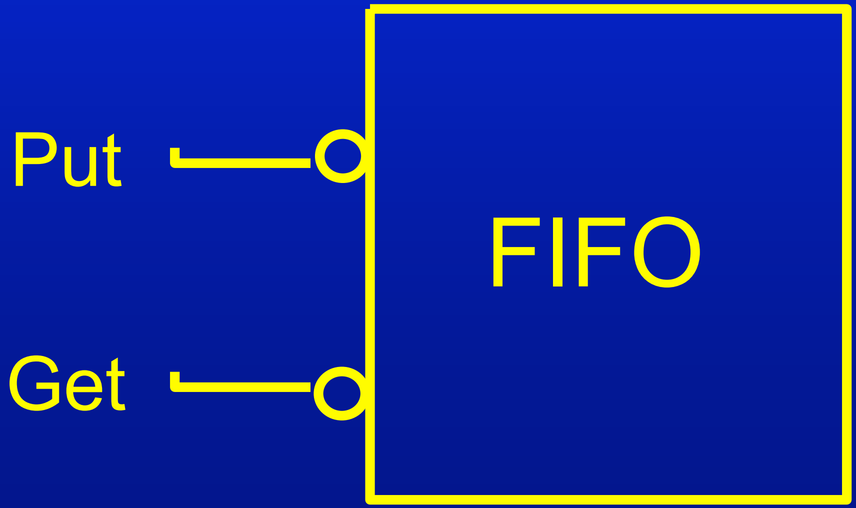
B.Coates, A.Davis, K.Stevens, *"The Post Office Experience: Designing a Large Asynchronous Chip"*, INTEGRATION: the VLSI Journal, vol. 15:3, pp. 341-66 (Oct. 1993)

Example #7: An Asynchronous FIFO

From Columbia University:

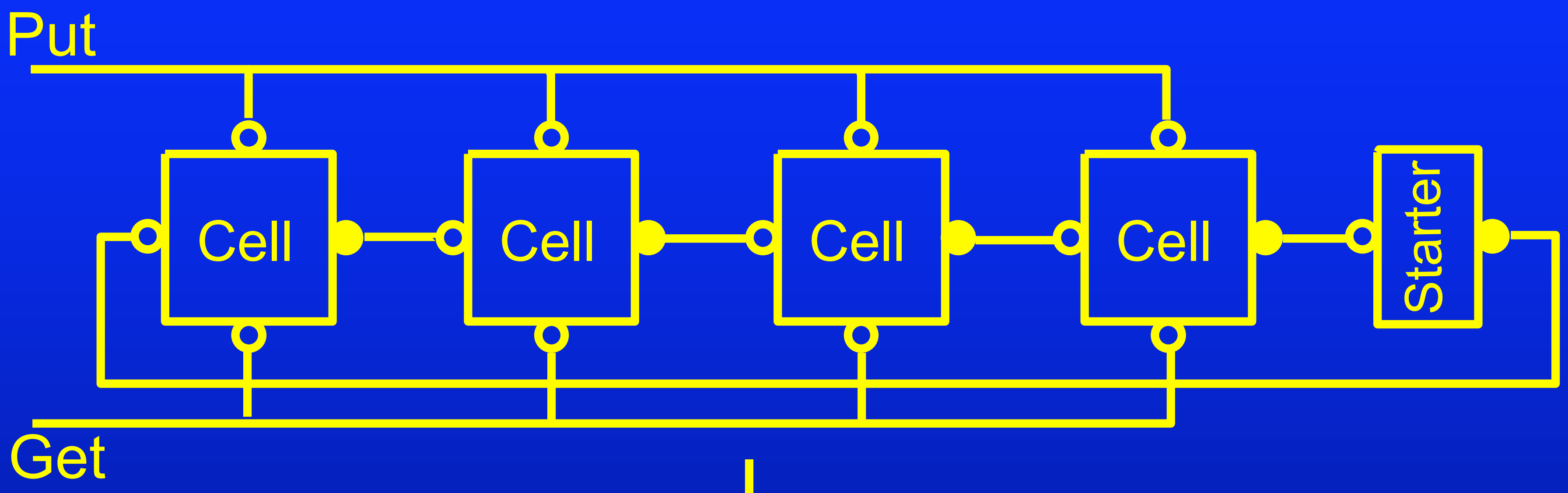
Tiberiu Chelcea and S. Nowick,
"Low-Latency Asynchronous FIFO's Using Token Rings",
IEEE Async-00 Symposium (Int. Sympos. on Advanced Research in Asynchronous Circuits & Systems), Eilat, Israel (April 2000)

Top-Level Block Diagram:

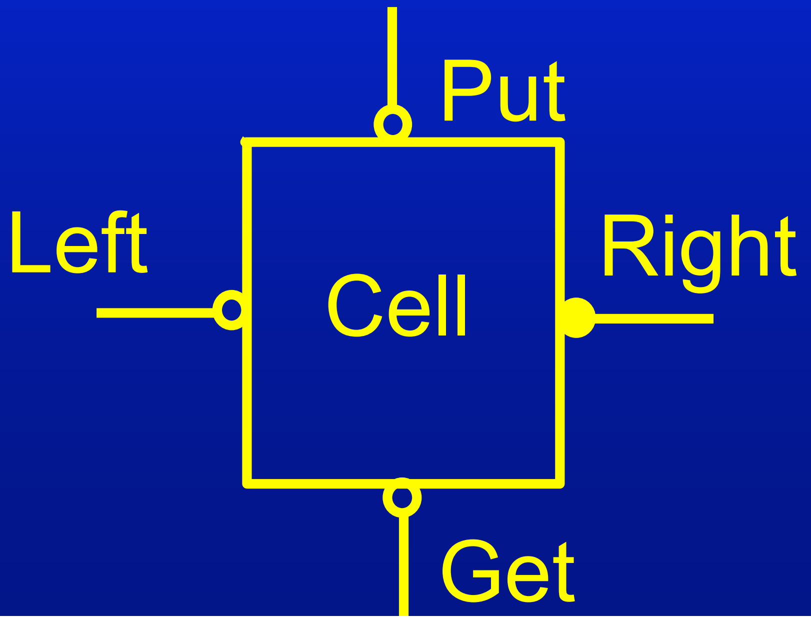


Example #7: An Async FIFO

FIFO: Token Ring Architecture

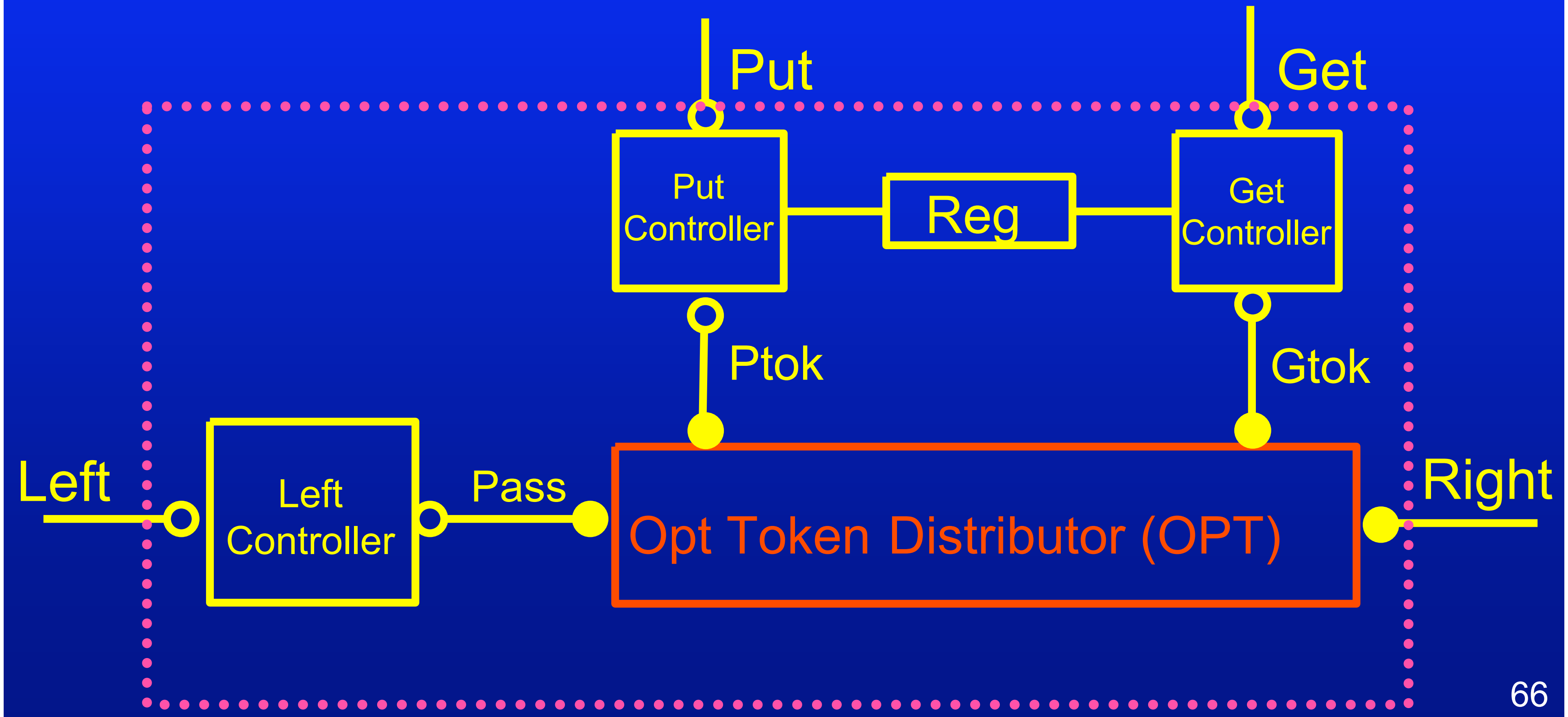


One FIFO Cell:



Example #7: An Async FIFO

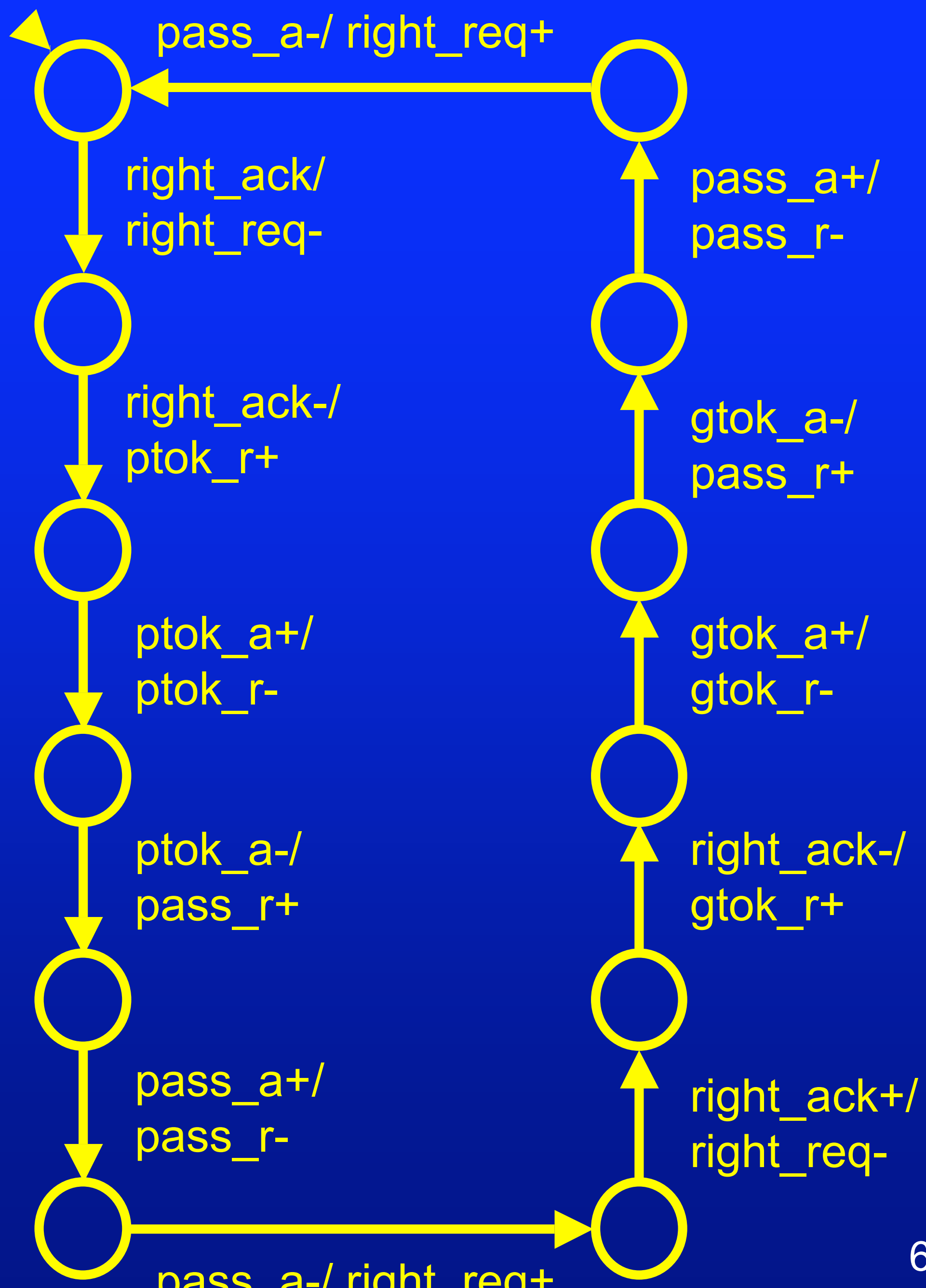
FIFO Cell: Decomposition



Example #7: Async FIFO

BM Spec:

"Opt Token Distributor" (OPT)



Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

1. MINIMALIST v2.0: installation and setup

Detailed instructions on installation can be found in the "INSTALL" and "README" files included with the release. This "INSTALL" file guides you through setting up your Linux "PATH" (and two other Linux variables, "MINIMALIST" and "LD_LIBRARY_PATH").

It also covers external tools required by MINIMALIST: ghostview, TCL interpreter, Perl interpreter, and dot. You need to have these external tools set up to use all the features of MINIMALIST. See instructions on how to download them, if you do not currently have them.

It also indicates the expected Linux environments on which MINIMALIST can be run.

If you still have problems with installation and setup, contact:
Steven Nowick (nowick@cs.columbia.edu). We will try to help.

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

2. Accessing "bm_decomp" and "MLO" tools

The MINIMALIST v2.0 release also includes two useful additional tools:
(i) "bm_decomp": a front-end pre-processor, which decomposes large BM specifications into several smaller interacting BM controllers; and
(ii) "MLO" (multi-level optimizer): a back-end post-processor, which converts two-level logic output to multi-level (Verilog format).

Neither tool is required: you can run the core MINIMALIST v2.0 package without using either (i) or (ii). However, both tools are useful, and each operates modularly as a "standalone" package (called from Linux shell).

To download these two tools, click on the MINIMALIST package at the CaSCADE web site (http://www1.cs.columbia.edu/~nowick/asynctools) and follow directions. Each of the tools comes with its own tutorial, examples and installation instructions.

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

3. Problems with graphical display:

MINIMALIST includes two tools for graphical display:

(i) "bms2ps" to display BM specifications (converts to Postscript); and

(ii) "plot_nand" to display BM implementations (converts to Postscript).

If you do not get a popup display window:

For example, if you are running MINIMALIST remotely (e.g. ssh) from a PC or Mac, consult details of your computer setup to enable it. If you still cannot get a popup window:

- use MINIMALIST to create Postscript (.ps) files for BM spec + implementation

  ==> both "bms2ps" and "plot_nand" always create .ps files
  (cleaner: to avoid error msgs., use "pla2nand")

- convert these .ps files to .pdf, then "ftp" them to your PC/Mac for display

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

3. Problems with graphical display (cont.):

If you do get a popup window, but no graphics appear:

You may have a font problem in your environment. The tools bms2ps, plot_nand and pla2nand all use Times and Helvetica Postscript fonts. In order to properly display the Postscript files produced by these tools, these fonts need to be installed on the system.

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

4. Reporting results: does the tool sometimes "miscount"?

When you run MINIMALIST on a BM specification, results for the two-level logic implementation are displayed: reporting "# products", "# literals", etc.

These results sometimes look wrong (it sometimes over-counts prods/lits)!

Explanation:

Consider an example: Z = A + BC.

The tool assumes a 'sum-of-products' (i.e. AND-OR) structure.

If a "product" (i.e. 'A') is just a wire (i.e. no AND gate needed) , the tool *still* counts it as one product (i.e. "AND1" gate).
It also counts one literal for this 'A' gate input, and one additional literal for the output of this "AND1" gate (i.e. as an input to the OR gate).

Summary: Minimalist sometimes reports worse results than actually obtained.

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

4. How well is "GC" (generalized C-element) mode supported?

MINIMALIST includes both scripts (minimalist-gc, minimalist-gc-suite) and command-line options (-G) to target generalized C-element implementations.

(a) extra (i.e. unused) state variables may be generated:

Sometimes the tool produces extra state variables in GC mode, which are not needed (I.e. feed into no logic). These can be discarded.

[The reason is that MINIMALIST currently uses the same 'state-min' step for GC as for non-GC runs, this step could be better optimized for GC in the future.]

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

4. How well is “GC” (generalized C-element) mode supported? (cont.)

(b) some features do not work with GC mode:

Several MINIMALIST commands do not support GC, or only indirectly:

- bms-verify: only works for 2-level circuits
- pla2verilog: currently no Verilog output for GC mode (for single GC cell)

  (but user can call pla2verilog to turn separate ‘set’ and ‘reset’ networks into Verilog)
- plot_nand, pla2nand: can only display separate ‘set’ or ‘reset’ networks, not an integrated single GC cell

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

5. Adding initialization circuitry: two-level vs. multi-level logic

MINIMALIST provides the "pla2verilog" option to (a) convert a two-level logic implementation to Verilog (.v file), and (b) insert initialization circuitry.

MINIMALIST also provides the "MLO" tool, to convert a two-level logic implementation to multi-level, and output Verilog (.v file).

A current restriction (v2.0) : "MLO" cannot be run after "pla2verilog". ("MLO" reads.pla/.sol files, not .v files.) So, if you want to insert initialization circuitry ("pla2verilog"), you must do so on the two-level circuit and then must skip multi-level optimization ("MLO").

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

6. Using scripts: does the ‘area script’ always produce the best area, etc.?

Not necessarily.

The ‘minimalist-area’/’minimalist-area-suite’ scripts are designed to tend to get better area (as observed on a number of examples). However, there is no guarantee! Sometimes, speed scripts will obtain better area than area scripts.

Similarly, the ‘minimalist-speed’/’minimalist-speed-suite’ scripts are designed to tend to get faster circuit speeds (as observed on a number of examples). However, sometimes an area’script will obtain better speed than speed scripts.

If you are concerned about finding the best possible implementation, try running several scripts and compare results. Also, try various custom runs in MinShell.

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

7. When running custom synthesis (in MinShell), any dangers?

Avoiding common errors: "no fedback output" vs. "fedback output":

There are three key BM synthesis steps *(in order)*:
(i) 'min-states', (ii) 'assign-states', and (iii) 'min-logic'.

Whether you are targeting a machine style "with fedback output" or "without fedback output", you must consistently set the '-F' flag for all three steps.

To synthesize an implementation with "fedback output":
- all three steps must be called with the '-F' flag

To synthesize an implementation with "no fedback output":
- all three steps must be called without the '-F' flag

NOTE: if you use the '-F' flag inconsistently in a run (i.e. use for some steps, not for others), the synthesis tool will produce an incorrect circuit!

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

8. Synthesizing larger controllers: how to reduce runtime of the tool?

See earlier slides in this tutorial on handling large examples, for some strategies to reduce the tool's runtime.

Another alternative is to *decompose* the original burst-mode specification, using the "bm_decomp" tool. See download site:

http://www1.cs.columbia.edu/~nowick/asynctools

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

9. How to get further help on various commands?

Type ‘help’ to see a list of Minimalist commands. Type ‘help’ on an individual command to get more details on usage, special modes, restrictions, and parameters.

Frequently-Asked Questions (FAQ)/ Clarifications on Features, etc.

10. How to find additional reading on MINIMALIST, burst-mode, etc.?

See the "README" file, which gives pointers to several readings. These in turn include many citations and leads on technical aspects of the tool (e.g. algorithms).

They also include pointers to some design case studies using burst-mode controllers (using earlier burst-mode synthesis tools), including:

- an experimental infrared communications chip (w/ HP Labs):

  A. Marshall, B. Coates and P. Siegel,
  "Designing an Asynchronous Communications Chip",
  IEEE Design & Test of Computers, vol. 11:2, pp. 8-21 (1994).

- a cache controller:

  S.M. Nowick, M.E. Dean, D.L. Dill and M. Horowitz,
  "The Design of a High-Performance Cache Controller: a Case Study in Asynchronous Synthesis."
  Integration: the VLSI Journal, vol. 15:3, pp. 241-262 (Oct. 1993).

- DRAM/SCSI controllers:

  S.M. Nowick, K.Y. Yun and D.L. Dill,
  "Practical Asynchronous Controller Design." Proc. of IEEE Int. Conf. on Computer Design,
  pp. 341-345 (Oct. 1992).