Copyright (C) 1994 by Alex Bronstein and Carolyn Talcott. All Rights Reserved.

You may copy and distribute verbatim copies of this Nqthm-1992 event script as you receive it, in any medium, including embedding it verbatim in derivative works, provided that you conspicuously and appropriately publish on each copy a valid copyright notice "Copyright (C) 1994 by Alex Bronstein and Carolyn Talcott. All Rights Reserved."

NO WARRANTY

Alex Bronstein and Carolyn Talcott PROVIDE ABSOLUTELY NO WARRANTY. THE EVENT SCRIPT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SCRIPT IS WITH YOU. SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT WILL Alex Bronstein or Carolyn Talcott BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

|#

EVENT: Start with the library "mlp" using the compiled version.

```
; bcd.bm: a BCD code checker, with serial (bit) input.
; . we have proved correctness w/ registers init to 0 (i.e.
; reality) AND for ARBITRARY init values ('r0,'r1); although it
; doesn't REALLY MEAN anything since the combinationals interpret
; that as ones... 2/6/89
; . we have also expressed the correctness hypothesis in various
; ways, to analyze idiom trade-offs for expressing that the
; "first 3 chars don't matter".
;
;;; CIRCUIT in SUGARED form:
```

#|

```
#|
(setq sysd '(sy-BCD (x)
(YO R 'rO x)
(Y1 R 'r1 Y0)
(Y2 S bor Y0 Y1)
(Yout S bnand x Y2)
))
(setq bcd '( |#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_bor.bm: Binary Or combinational element
; U7-DONE
DEFINITION:
bor(u, v)
= if (u = 0) \land (v = 0) then 0
     else 1 endif
; Everything below generated by: (bmcomb 'bor '() '(x y))
DEFINITION:
s-bor (x, y)
= if empty (x) then E
     else a (s-bor (p(x), p(y)), bor (l(x), l(y))) endif
;; A2-Begin-S-BOR
THEOREM: a2-empty-s-bor
\operatorname{empty}(\operatorname{s-bor}(x, y)) = \operatorname{empty}(x)
THEOREM: a2-e-s-bor
(s-bor(x, y) = E) = empty(x)
THEOREM: a2-lp-s-bor
\ln\left(\text{s-bor}\left(x, \, y\right)\right) = \ln\left(x\right)
THEOREM: a2-lpe-s-bor
eqlen (s-bor (x, y), x)
Theorem: a2-ic-s-bor
(\operatorname{len}(x) = \operatorname{len}(y))
 \rightarrow (\operatorname{s-bor}(\operatorname{i}(c_x, x), \operatorname{i}(c_y, y)) = \operatorname{i}(\operatorname{bor}(c_x, c_y), \operatorname{s-bor}(x, y)))
```

THEOREM: a2-lc-s-bor $(\neg \operatorname{empty}(x)) \rightarrow (\operatorname{l}(\operatorname{s-bor}(x, y)) = \operatorname{bor}(\operatorname{l}(x), \operatorname{l}(y)))$ THEOREM: a2-pc-s-bor p(s-bor(x, y)) = s-bor(p(x), p(y))THEOREM: a2-hc-s-bor $\left(\left(\neg \operatorname{empty}\left(x\right)\right) \land \left(\operatorname{len}\left(x\right) = \operatorname{len}\left(y\right)\right)\right)$ $\rightarrow \quad (h(s-bor(x, y)) = bor(h(x), h(y)))$ THEOREM: a2-bc-s-bor $(\operatorname{len}(x) = \operatorname{len}(y)) \rightarrow (\operatorname{b}(\operatorname{s-bor}(x, y)) = \operatorname{s-bor}(\operatorname{b}(x), \operatorname{b}(y)))$ THEOREM: a2-bnc-s-bor $(\operatorname{len}(x) = \operatorname{len}(y)) \to (\operatorname{bn}(n, \operatorname{s-bor}(x, y)) = \operatorname{s-bor}(\operatorname{bn}(n, x), \operatorname{bn}(n, y)))$;; A2-End-S-BOR ; eof:comb_bor.bm ; comb_bnand.bm: Binary Nand combinational element ; U7-DONE **DEFINITION:** bnand (u, v)= if $(u = 0) \lor (v = 0)$ then 1 else 0 endif ; Everything below generated by: (bmcomb 'bnand '() '(x y)) **DEFINITION:** s-bnand (x, y)= **if** empty (x) **then** E else a (s-bnand (p(x), p(y)), bnand (l(x), l(y))) endif ;; A2-Begin-S-BNAND THEOREM: a2-empty-s-bnand empty(s-bnand(x, y)) = empty(x)THEOREM: a2-e-s-bnand (s-bnand(x, y) = E) = empty(x)

THEOREM: a2-lp-s-bnand len (s-bnand (x, y)) = len (x)

THEOREM: a2-lpe-s-bnand eqlen (s-bnand (x, y), x)

THEOREM: a2-ic-s-bnand $(\operatorname{len}(x) = \operatorname{len}(y))$ \rightarrow (s-bnand (i (*c_x*, *x*), i (*c_y*, *y*)) = i (bnand (*c_x*, *c_y*), s-bnand (*x*, *y*)))

THEOREM: a2-lc-s-bnand $(\neg \text{ empty } (x)) \rightarrow (l(\text{s-bnand } (x, y)) = \text{bnand } (l(x), l(y)))$

THEOREM: a2-pc-s-bnand p(s-bnand(x, y)) = s-bnand(p(x), p(y))

THEOREM: a2-hc-s-bnand $((\neg \text{ empty } (x)) \land (\text{len } (x) = \text{len } (y)))$ $\rightarrow \quad (h (s-\text{bnand } (x, y)) = \text{bnand } (h (x), h (y)))$

THEOREM: a2-bc-s-bnand (len (x) = len(y)) \rightarrow (b (s-bnand (x, y)) = s-bnand (b (x), b (y)))

THEOREM: a2-bnc-s-bnand $(\operatorname{len}(x) = \operatorname{len}(y)) \rightarrow (\operatorname{bn}(n, \operatorname{s-bnand}(x, y)) = \operatorname{s-bnand}(\operatorname{bn}(n, x), \operatorname{bn}(n, y)))$

;; A2-End-S-BNAND

; eof:comb_bnand.bm

DEFINITION: topor-sy-bcd (ln)= if ln = 'y0 then 0 elseif ln = 'y1 then 0 elseif ln = 'y2 then 1 elseif ln = 'yout then 2 else 0 endif

DEFINITION: sy-bcd (ln, x)= if ln = 'y0then if empty (x) then E else i ('r0, p (x)) endif elseif ln = 'y1

```
then if empty(x) then E
          else i ('r1, sy-bcd ('y0, p(x))) endif
    elseif ln = 'y^2 then s-bor (sy-bcd ('y0, x), sy-bcd ('y1, x))
    elseif ln = 'yout then s-bnand (x, \text{ sy-bcd}('y2, x))
    else sfix (x) endif
;; A2-Begin-SY-BCD
THEOREM: a2-empty-sy-bcd
empty(sy-bcd(ln, x)) = empty(x)
THEOREM: a2-e-sy-bcd
(\text{sy-bcd}(ln, x) = E) = \text{empty}(x)
THEOREM: a2-lp-sy-bcd
\ln\left(\text{sy-bcd}\left(ln,\,x\right)\right) = \ln\left(x\right)
THEOREM: a2-lpe-sy-bcd
eqlen (sy-bcd (ln, x), x)
THEOREM: a2-pc-sy-bcd
p(sy-bcd(ln, x)) = sy-bcd(ln, p(x))
;; A2-End-SY-BCD
;;; Circuit CORRECTNESS /Paillet:
; BCD-bits defines a correct binary coded decimal,
      b0 is most-significant.
:
DEFINITION:
bcd-bits (b0, b1, b2, b3) = ((b0 = 0) \lor ((b1 = 0) \land (b2 = 0)))
; BCD-Spec defines what the circuit is supposed to compute, by its
; last char.
DEFINITION:
bcd-spec(x) = bcd-bits(l(x), l(p(x)), l(p(p(x))), l(p(p(x)))))
; CORRECTNESS:
; original correctness statement:
THEOREM: bcd-correct
((\neg \operatorname{empty}(x)))
 \land \quad (\neg \text{ empty} (\mathbf{p} (x)))
 \land \quad (\neg \text{ empty} (p (p (x))))
 \land \quad (\neg \text{ empty} (p (p (p (x))))))
```

 \rightarrow (bibo (l(sy-bcd('yout, x))) = bcd-spec(x))

```
; alternative: implicitely using P(n+1) x /=e => P(n) x,...,Px,x /=e
    ALSO works, with many more time and more CASES because BM
    explores all the "obviously impossible" cases and takes some
    time reducing the hyp to F.
THEOREM: bcd-correct2
(\neg \text{ empty} (p(p(x))))) \rightarrow (\text{bibo} (l(\text{sy-bcd}(\text{'yout}, x))) = \text{bcd-spec}(x))
; alternative: use the Pn operator instead of explicit P's in hyp:
    ALSO works, with same number of CASES as explicit P's, but a
    little bit more time due to the expansions of Pn required.
:
THEOREM: bcd-correct3
(\neg \text{ empty}(\operatorname{pn}(3, x))) \rightarrow (\operatorname{bibo}(\operatorname{l}(\operatorname{sy-bcd}(\operatorname{'yout}, x))) = \operatorname{bcd-spec}(x))
; alternative: use LENgth > 3 instead of Pn or explicit P's.
    ALSO works, with same number of CASES and time as Pn
THEOREM: bcd-correct4
(3 < \text{len}(x)) \rightarrow (\text{bibo}(l(\text{sy-bcd}(\text{yout}, x))) = \text{bcd-spec}(x))
;; one can give a CONTRIVED reading of Paillet's as pure streams:
DEFINITION:
bcd-bits-s (b0, b1, b2, b3)
= s-or (s-equal (b\theta, s-const(0, b\theta))),
         s-and (s-equal (b1, s-const(0, b1)), s-equal (b2, s-const(0, b2))))
DEFINITION:
bcd-spec-s (x) = bcd-bits-s(x, p(x), p(p(x)), p(p(p(x))))
; but do NOT have the equality Yout = BCD-Spec-S because of the
; initial 3 values output by the circuit. To get around that, we
; could:
      - redefine BCD-Spec-S to be a big IF statement on the length
        of x, and put in the explicit values computed by the
        circuit, obtained by evaluating SY-BCD in rloop.
        [ A major pain ]
     - somehow express that we're not looking at the first 3 values
:
        output.
;
        We could use (RST (RST (RST ... ) = (RST (RST (RST ... ) but
        clearly that would not work, since we currently have nothing
        about RST.
; Note that this is essentially the PIPELINE problem.
; So far, the best approach seems to talk about L(sysd) which is
```

; most natural. ; NOTE: now that we've defined B (instead of RST) to deal with ; pipelines, and entered its theory, the above discussioon is moot.

; eof: bcd.bm
;))

Index

a, 2, 3 a2-bc-s-bnand, 4 a2-bc-s-bor, 3 a2-bnc-s-bnand, 4 a2-bnc-s-bor, 3 a2-e-s-bnand, 3 a2-e-s-bor, 2 a2-e-sy-bcd, 5 a2-empty-s-bnand, 3 a2-empty-s-bor, 2 a2-empty-sy-bcd, 5 a2-hc-s-bnand, 4 a2-hc-s-bor, 3 a2-ic-s-bnand, 4 a2-ic-s-bor, 2 a2-lc-s-bnand, 4 a2-lc-s-bor, 3 a2-lp-s-bnand, 4 a2-lp-s-bor, 2 a2-lp-sy-bcd, 5 a2-lpe-s-bnand, 4 a2-lpe-s-bor, 2 a2-lpe-sy-bcd, 5 a2-pc-s-bnand, 4 a2-pc-s-bor, 3 a2-pc-sy-bcd, 5

b, 3, 4 bcd-bits, 5 bcd-bits-s, 6 bcd-correct, 5

bcd-correct2, 6 bcd-correct3, 6 bcd-correct4, 6 bcd-spec, 5, 6 bcd-spec-s, 6 bibo, 5, 6 bn, 3, 4 bnand, 3, 4 bor, 2, 3 e, 2–5 empty, 2–6 eqlen, 2, 4, 5 h, 3, 4 i, 2, 4, 5 1, 2–6 len, 2-6p, 2–6 pn, 6 s-and, 6 s-bnand, 3–5 s-bor, 2, 3, 5 s-const, 6 s-equal, 6 s-or, 6sfix, 5sy-bcd, 4-6

topor-sy-bcd, 4