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)
(Y0 R 'r0 x)
(Y1 R 'r1 Y0)
(Y2 S bor Y0 Y1)
(Yout S bandal x Y2))
)

(setq bcd '()

; BM DEFINITIONS and A2 LEMMAS, generated by BMPSYSD:
; comb_bor.bm: Binary Or combinational element
; U7-DONE

DEFINITION:
bor (u, v)
  = if (u = 0) ∧ (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
empty (s-bor (x, y)) = empty (x)

THEOREM: a2-e-s-bor
(s-bor (x, y) = E) = empty (x)

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

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

THEOREM: a2-ic-s-bor
(len (x) = len (y))
→ (s-bor (i (x, x), i (y, y)) = i (bor (x, c y), s-bor (x, y)))
**Theorem: a2-lc-s-bor**

\(\neg \text{empty}(x) \rightarrow (1\text{-bor}(x, y)) = \text{bor}(1(x), 1(y))\)

**Theorem: a2-pc-s-bor**

\(p(s\text{-bor}(x, y)) = s\text{-bor}(p(x), p(y))\)

**Theorem: a2-hc-s-bor**

\(((\neg \text{empty}(x)) \land (\text{len}(x) = \text{len}(y))) \rightarrow (h(s\text{-bor}(x, y)) = \text{bor}(h(x), h(y)))\)

**Theorem: a2-bc-s-bor**

\((\text{len}(x) = \text{len}(y)) \rightarrow (b(s\text{-bor}(x, y)) = s\text{-bor}(b(x), b(y)))\)

**Theorem: a2-bnc-s-bor**

\((\text{len}(x) = \text{len}(y)) \rightarrow (\text{bn}(n, s\text{-bor}(x, y)) = s\text{-bor}(\text{bn}(n, x), \text{bn}(n, y)))\)

;; A2-End-S-BOR

; eof:comb_bor.bm

; comb_bnand.bm: Binary Nand combinational element
; U7-DONE

**Definition:**

\(\text{bnand}(u, v) = \begin{cases} 
1 & \text{if } (u = 0) \lor (v = 0) \\
0 & \text{else}
\end{cases}\)

; Everything below generated by: (bmcomb 'bnand () '(x y))

**Definition:**

\(s\text{-bnand}(x, y) = \begin{cases} 
\text{E} & \text{if } \text{empty}(x) \\
\text{a}(s\text{-bnand}(p(x), p(y)), \text{bnand}(1(x), 1(y))) & \text{else}
\end{cases}\)

;; A2-Begin-S-BNAND

**Theorem: a2-empty-s-bnand**

\(\text{empty}(s\text{-bnand}(x, y)) = \text{empty}(x)\)

**Theorem: a2-e-s-bnand**

\((s\text{-bnand}(x, y) = \text{E}) = \text{empty}(x)\)
Theorem: a2-lp-s-bnand
\[ \text{len} (s\text{-bnand} (x, y)) = \text{len} (x) \]

Theorem: a2-lpe-s-bnand
eqlen (s\text{-bnand} (x, y), x)

Theorem: a2-ic-s-bnand
\[(\text{len} (x) = \text{len} (y)) \rightarrow (s\text{-bnand} (i (c\_x, x), i (c\_y, y)) = i (\text{bnand} (c\_x, c\_y), s\text{-bnand} (x, y)))\]

Theorem: a2-lc-s-bnand
\[ (\neg \text{empty} (x)) \rightarrow (1 \text{-s-bnand} (x, y)) = \text{bnand} (1 (x), 1 (y)) \]

Theorem: a2-pc-s-bnand
\[ p (s\text{-bnand} (x, y)) = s\text{-bnand} (p (x), p (y)) \]

Theorem: a2-hc-s-bnand
\[(\neg \text{empty} (x)) \land (\text{len} (x) = \text{len} (y)) \rightarrow (h (s\text{-bnand} (x, y)) = \text{bnand} (h (x), h (y)))\]

Theorem: a2-bc-s-bnand
\[(\text{len} (x) = \text{len} (y)) \rightarrow (b (s\text{-bnand} (x, y)) = s\text{-bnand} (b (x), b (y)))\]

Theorem: a2-bnc-s-bnand
\[(\text{len} (x) = \text{len} (y)) \rightarrow (b (n, s\text{-bnand} (x, y)) = s\text{-bnand} (b (n, x), b (n, y)))\]

;; A2-End-S-BNAND

; eof:comb_bnand.bm

Definition:
topor-sy-bcd \[(ln) \]
\[= \text{if } ln = 'y0 \text{ then } 0 \text{ elseif } ln = 'y1 \text{ then } 0 \text{ elseif } ln = 'y2 \text{ then } 1 \text{ elseif } ln = 'yout \text{ then } 2 \text{ else } 0 \text{ endif}\]

Definition:
sy-bcd \[(ln, x) \]
\[= \text{if } ln = 'y0 \text{ then if } \text{empty} (x) \text{ then } \text{E } \text{else } i ('r0, p (x)) \text{ endif } \text{elseif } ln = 'y1 \text{ endif}\]
then if empty(x) then E
   else i(r1, sy-bcd(y0, p(x))) endif
elseif ln = 'y2 then s-bor(sy-bcd(y0, x), sy-bcd(y1, x))
elseif ln = 'yout then s-bnand(x, 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
(sy-bcd(ln, x) = E) = empty(x)

THEOREM: a2-lp-sy-bcd
len(sy-bcd(ln, x)) = len(x)

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) ∨ ((b1 = 0) ∧ (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(p(x)))))

; CORRECTNESS:
; original correctness statement:

THEOREM: bcd-correct
((¬ empty(x))
 ∧ (¬ empty(p(x)))
 ∧ (¬ empty(p(p(x))))
 ∧ (¬ empty(p(p(p(x)))))
→ (bibo(l(sy-bcd('yout, x))) = bcd-spec(x))
Theorem: bcd-correct2
\( \neg \text{empty}(p(p(p(x)))) \rightarrow (\text{bibo}(l(\text{sy-bcd('yout, x)))) = \text{bcd-spec}(x)) \)

; 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
\( \neg \text{empty}(pn(3, x)) \rightarrow (\text{bibo}(l(\text{sy-bcd('yout, x)))) = \text{bcd-spec}(x)) \)

; alternative: use \( \text{LENgth} > 3 \) instead of \( Pn \) or explicit \( P \)'s.
; ALSO works, with same number of CASES and time as \( Pn \)

Theorem: bcd-correct4
\( 3 < \text{len}(x) \rightarrow (\text{bibo}(l(\text{sy-bcd('yout, x)))) = \text{bcd-spec}(x)) \)

;; one can give a CONTRIVED reading of Paillet's as pure streams:

Definition:
\[ \text{bcd-bits-s}(b0, b1, b2, b3) = \text{s-or}(\text{s-equal}(b0, \text{s-const}(0, b0)), \text{s-and}(\text{s-equal}(b1, \text{s-const}(0, b1)), \text{s-equal}(b2, \text{s-const}(0, b2)))) \]

Definition:
\[ \text{bcd-spec-s}(x) = \text{bcd-bits-s}(x, p(x), p(p(x)), p(p(p(x)))) \]

; but do NOT have the equality \( \text{Yout} = \text{BCD-Spec-S} \) because of the
; initial 3 values output by the circuit. To get around that, we
; could:
; - redefine \( \text{BCD-Spec-S} \) to be a big IF statement on the length
;   of \( x \), and put in the explicit values computed by the
;   circuit, obtained by evaluating \( \text{SY-BCD} \) in rloop.
; [ A major pain ]
; - somehow express that we're not looking at the first 3 values
; output.
; We could use \( \text{RST (RST (RST ... }} \) = \( \text{RST (RST (RST ... } \) but
; clearly that would not work, since we currently have nothing
; about \( \text{RST} \).
; Note that this is essentially the PIPELINE problem.
; So far, the best approach seems to talk about \( L(\text{sysd}) \) which is
; most natural.
; NOTE: now that we've defined B (instead of RST) to deal with
; pipelines, and entered its theory, the above discussion 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-le-s-bnand, 4
a2-le-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

l, 2–6
len, 2–6

p, 2–6
pn, 6

s-and, 6
s-bnand, 3–5
s-bor, 2, 3, 5
s-const, 6
s-equal, 6
s-or, 6
sfix, 5
sy-bcd, 4–6
topor-sy-bcd, 4