```
#|
```



```
|#
```

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

```
; counterR.bm: a resetable clock counter, i.e. Paillet example 1.
;
; This is our first circuit w/ 2 inputs, yet to my amazement,
; extra EQLEN hypotheses were NEVER required, neither in the A2
; lemmas, nor in the actual specifications (Paillet, or mine)!  Of
; course, when they became required (in Paillet #7) and put in
; Sugar, they were installed here.
;
; NOTE (w/ EMPTY enabled) : Attempts to speed up A2-PC failed
; miserably: disabling the combinationals fails because of the
; uneven cases, and the fact that BM can't derive (empty y) from
```

```
; (empty x) and equal (len x) (len y).
; A problem we have had for a long time, and which will probably
; persist until we solve EQLEN.
; Giving the induction hint ahead of time made things worse, as
; usual. Twiddling with STR-P-I2 improved nothing.



;;; CIRCUIT in SUGARED form:
#|
(setq sysd '(sy-COUNT (Xc Xe)
(Ymux  S Mux Xc Xe Yinc)
(Yreg  R 0 YMux)
(Yinc  S Incn Yreg)
))

(setq counterR '( |#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_mux.bm: Mux combinational element, i.e. "if".
; U7-DONE
```

DEFINITION:
mux (*u1*, *u2*, *u3*)
= **if** *u1* **then** *u2*
else *u3* **endif**

```
; everything below generated by: (bmcomb 'mux '() '(x1 x2 x3))
; with the EXCEPTIONS/HAND-MODIFICATIONS given below.
```

DEFINITION:
s-mux (*x1*, *x2*, *x3*)
= **if** empty (*x1*) **then** E
else a (s-mux (p (*x1*), p (*x2*), p (*x3*)), mux (l (*x1*), l (*x2*), l (*x3*))) **endif**

```
; SMUX-is-SIF can make things much simpler on occasions:
```

THEOREM: smux-is-sif
s-mux (*x1*, *x2*, *x3*) = s-if (*x1*, *x2*, *x3*)

EVENT: Disable smux-is-sif.

```
; We take advantage of SMUX-is-SIF for all inductive proofs. To do so we
```

```
; HAND-MODIFY the code generated by Sugar to replace all the hints by
;   - A2-EMPTY, A2-PC replace hint with: ((enable smux-is-sif))
;   - A2-LP, A2-IC, A2-HC, A2-BC: ((enable smux-is-sif) (disable len))
;   - A2-BNC: ((enable smux-is-sif) (disable bn len))

;; A2-Begin-S-MUX
```

THEOREM: a2-empty-s-mux
empty (s-mux ($x1$, $x2$, $x3$)) = empty ($x1$)

THEOREM: a2-e-s-mux
(s-mux ($x1$, $x2$, $x3$) = E) = empty ($x1$)

THEOREM: a2-lp-s-mux
len (s-mux ($x1$, $x2$, $x3$)) = len ($x1$)

THEOREM: a2-lpe-s-mux
eqlen (s-mux ($x1$, $x2$, $x3$), $x1$)

THEOREM: a2-ic-s-mux
((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$)))
$\rightarrow$ (s-mux (i (c_x1, $x1$), i (c_x2, $x2$), i (c_x3, $x3$))
= i (mux (c_x1, c_x2, c_x3), s-mux ($x1$, $x2$, $x3$)))

THEOREM: a2-lc-s-mux
($\neg$ empty ($x1$)) $\rightarrow$ (l (s-mux ($x1$, $x2$, $x3$)) = mux (l ($x1$), l ($x2$), l ($x3$)))

THEOREM: a2-pc-s-mux
p (s-mux ($x1$, $x2$, $x3$)) = s-mux (p ($x1$), p ($x2$), p ($x3$))

THEOREM: a2-hc-s-mux
(($\neg$ empty ($x1$)) $\land$ ((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$))))
$\rightarrow$ (h (s-mux ($x1$, $x2$, $x3$)) = mux (h ($x1$), h ($x2$), h ($x3$)))

```
;old:    ((DISABLE MUX S-MUX) (ENABLE H LEN) (INDUCT (S-MUX X1 X2 X3)))
```

THEOREM: a2-bc-s-mux
((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$)))
$\rightarrow$ (b (s-mux ($x1$, $x2$, $x3$)) = s-mux (b ($x1$), b ($x2$), b ($x3$)))

```
;old:    ((DISABLE MUX) (ENABLE B LEN) (INDUCT (S-MUX X1 X2 X3)))
```

THEOREM: a2-bnc-s-mux
((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$)))
$\rightarrow$ (bn (n, s-mux ($x1$, $x2$, $x3$)) = s-mux (bn (n, $x1$), bn (n, $x2$), bn (n, $x3$)))

```
;old: ((DISABLE MUX S-MUX))

;; A2-End-S-MUX

; eof:comb_mux.bm

; comb_incn.bm: Inc modulo N combinational element, a minor modification
;               of comb_inc which shouldn't cause any difference, unless
;               the loop-around property is used in a critical way, which is
;               rare.
; Note that N is treated as a global constant, but not as an individual
; parameter, so we don't have to carry it around everywhere.  This is just
; an experiment, to see what's more convenient.
; U7-DONE
```

EVENT: Introduce the function symbol n of 0 arguments.

```
; we may want to add an axiom saying that it's a number, not needed so far..
```

DEFINITION:
incn (u)
= **if** $u = \text{N}$ **then** 0
else $1 + u$ **endif**

```
; Everything below generated by: (bmcomb 'incn '() '(x))
```

DEFINITION:
s-incn (x)
= **if** empty (x) **then** E
else a (s-incn (p (x)), incn (l (x))) **endif**

```
;; A2-Begin-S-INCN
```

THEOREM: a2-empty-s-incn
empty (s-incn (x)) = empty (x)

THEOREM: a2-e-s-incn
(s-incn (x) = E) = empty (x)

THEOREM: a2-lp-s-incn
len (s-incn (x)) = len (x)

THEOREM: a2-lpe-s-incn
eqlen (s-incn (x), x)

THEOREM: a2-ic-s-incn
s-incn (i (c_x, x)) = i (incn (c_x), s-incn (x))

THEOREM: a2-lc-s-incn
($\neg$ empty (x)) $\rightarrow$ (l (s-incn (x)) = incn (l (x)))

THEOREM: a2-pc-s-incn
p (s-incn (x)) = s-incn (p (x))

THEOREM: a2-hc-s-incn
($\neg$ empty (x)) $\rightarrow$ (h (s-incn (x)) = incn (h (x)))

THEOREM: a2-bc-s-incn
b (s-incn (x)) = s-incn (b (x))

THEOREM: a2-bnc-s-incn
bn (n, s-incn (x)) = s-incn (bn (n, x))

```
;; A2-End-S-INCN

; eof:comb_incn.bm
```

DEFINITION:
topor-sy-count (*ln*)
= **if** *ln* = `'ymux` **then** `2`
elseif *ln* = `'yreg` **then** `0`
elseif *ln* = `'yinc` **then** `1`
else `0` **endif**

DEFINITION:
sy-count (*ln*, *xc*, *xe*)
= **if** *ln* = `'ymux` **then** s-mux (*xc*, *xe*, sy-count (`'yinc`, *xc*, *xe*))
elseif *ln* = `'yreg`
then if empty (*xc*) **then** E
else i (`0`, sy-count (`'ymux`, p (*xc*), p (*xe*))) **endif**
elseif *ln* = `'yinc` **then** s-incn (sy-count (`'yreg`, *xc*, *xe*))
else sfix (*xc*) **endif**

```
;; A2-Begin-SY-COUNT
```

THEOREM: a2-empty-sy-count
$(\text{len}(xc) = \text{len}(xe)) \rightarrow (\text{empty}(\text{sy-count}(ln, xc, xe)) = \text{empty}(xc))$

THEOREM: a2-e-sy-count
$(\text{len}(xc) = \text{len}(xe)) \rightarrow ((\text{sy-count}(ln, xc, xe) = \text{E}) = \text{empty}(xc))$

THEOREM: a2-lp-sy-count
$(\text{len}(xc) = \text{len}(xe)) \rightarrow (\text{len}(\text{sy-count}(ln, xc, xe)) = \text{len}(xc))$

THEOREM: a2-lpe-sy-count
$(\text{len}(xc) = \text{len}(xe)) \rightarrow \text{eqlen}(\text{sy-count}(ln, xc, xe), xc)$

THEOREM: a2-pc-sy-count
$(\text{len}(xc) = \text{len}(xe))$
$\rightarrow \quad (\text{p}(\text{sy-count}(ln, xc, xe)) = \text{sy-count}(ln, \text{p}(xc), \text{p}(xe)))$

```
;; A2-End-SY-COUNT

;;; Circuit CORRECTNESS /Paillet:

; Note that as originally stated in Paillet, with the P outside of
; sy-count makes for a looping (unfolding) which would have to be
; proved kludgeily, and would be useless.  The following rule can
; be used as a rewrite.
```

THEOREM: count-paillet-correct
$((\neg\ \text{empty}(xc)) \wedge (\neg\ \text{empty}(xe)))$
$\rightarrow \quad (\text{sy-count}(\texttt{'yreg}, xc, xe)$
$\quad\quad = \quad \text{i}(\texttt{0}, \text{s-if}(\text{p}(xc), \text{p}(xe), \text{s-incn}(\text{sy-count}(\texttt{'yreg}, \text{p}(xc), \text{p}(xe))))))$

```
; The "last-char" reading of the spec yields:
; NOTE: we can prove it by repeating the same hint and disabling
; CORRECT, i.e. independently.  Trying to use CORRECT fails
; miserably because it also triggers on:
; (sy-count 'Yreg (P Xc) (P Xe)).  Note also that we need the
; EQ-LEN hyp because we need A2-EMPTY-SY-COUNT.
```

THEOREM: count-paillet-correct-l
$((\neg\ \text{empty}(\text{p}(xc))) \wedge (\neg\ \text{empty}(\text{p}(xe))) \wedge (\text{len}(xc) = \text{len}(xe)))$
$\rightarrow \quad (\text{l}(\text{sy-count}(\texttt{'yreg}, xc, xe))$
$\quad\quad = \quad \textbf{if}\ \text{l}(\text{p}(xc))\ \ \textbf{then}\ \text{l}(\text{p}(xe))$
$\quad\quad\quad\quad \textbf{else}\ \text{incn}(\text{l}(\text{sy-count}(\texttt{'yreg}, \text{p}(xc), \text{p}(xe))))\ \textbf{endif})$

```
; eof: counterR.bm
;))
```

Index