```
#|
```



```
|#
```

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

```
; handrec.bm: a HANDSHAKE RECEIVER piece, Paillet #6
;
;  Note: The same remark as for Serial applies: we were able to get
;  this whole multi-input circuit through without EQ-LEN hyps, but
;  that's because of a lucky shot in line Y4 (order made right).
;  The more general version is of course slower and uglier...
;

;;; CIRCUIT in SUGARED form:
#|
(setq sysd '(sy-HANDREC (Xcall Xm)
```

```
(Yrcall R F Xcall)
(Y0 S not Yrcall)
(Y1 S and Yrcall Yhear)
(Y2 S or Y0 Y1)
(Y3 S and Xcall Y2)
(Y4 S and Xcall Xm)
(Y5 S and Y4 Y0)
(Yhear R F Y3)
(Yinfin R F Y5)
))

(setq handrec '( |#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
```

DEFINITION:
topor-sy-handrec (*ln*)
= **if** *ln* = `'yrcall` **then** `0`
  elseif *ln* = `'y0` **then** `1`
  elseif *ln* = `'y1` **then** `1`
  elseif *ln* = `'y2` **then** `2`
  elseif *ln* = `'y3` **then** `3`
  elseif *ln* = `'y4` **then** `1`
  elseif *ln* = `'y5` **then** `2`
  elseif *ln* = `'yhear` **then** `0`
  elseif *ln* = `'yinfin` **then** `0`
  else `0` **endif**

DEFINITION:
sy-handrec (*ln*, *xcall*, *xm*)
= **if** *ln* = `'yrcall`
  then if empty (*xcall*) **then** E
    else i (**f**, p (*xcall*)) **endif**
  elseif *ln* = `'y0` **then** s-not (sy-handrec (`'yrcall`, *xcall*, *xm*))
  elseif *ln* = `'y1`
  then s-and (sy-handrec (`'yrcall`, *xcall*, *xm*),
    sy-handrec (`'yhear`, *xcall*, *xm*))
  elseif *ln* = `'y2`
  then s-or (sy-handrec (`'y0`, *xcall*, *xm*), sy-handrec (`'y1`, *xcall*, *xm*))
  elseif *ln* = `'y3` **then** s-and (*xcall*, sy-handrec (`'y2`, *xcall*, *xm*))
  elseif *ln* = `'y4` **then** s-and (*xcall*, *xm*)
  elseif *ln* = `'y5`
  then s-and (sy-handrec (`'y4`, *xcall*, *xm*), sy-handrec (`'y0`, *xcall*, *xm*))
  elseif *ln* = `'yhear`

then if empty ($xcall$) **then** E
else i (**f**, sy-handrec (`'y3`, p ($xcall$), p (xm))) **endif**
elseif ln = `'yinfin`
then if empty ($xcall$) **then** E
else i (**f**, sy-handrec (`'y5`, p ($xcall$), p (xm))) **endif**
else sfix ($xcall$) **endif**

```
;; A2-Begin-SY-HANDREC
```

THEOREM: a2-empty-sy-handrec
(len ($xcall$) = len (xm))
$\rightarrow$ (empty (sy-handrec (ln, $xcall$, xm)) = empty ($xcall$))

THEOREM: a2-e-sy-handrec
(len ($xcall$) = len (xm))
$\rightarrow$ ((sy-handrec (ln, $xcall$, xm) = E) = empty ($xcall$))

THEOREM: a2-lp-sy-handrec
(len ($xcall$) = len (xm)) $\rightarrow$ (len (sy-handrec (ln, $xcall$, xm)) = len ($xcall$))

THEOREM: a2-lpe-sy-handrec
(len ($xcall$) = len (xm)) $\rightarrow$ eqlen (sy-handrec (ln, $xcall$, xm), $xcall$)

THEOREM: a2-pc-sy-handrec
(len ($xcall$) = len (xm))
$\rightarrow$ (p (sy-handrec (ln, $xcall$, xm)) = sy-handrec (ln, p ($xcall$), p (xm)))

```
;; A2-End-SY-HANDREC

;;; Circuit CORRECTNESS /Paillet:

; The first correctness formula derived from the temporal logic spec
; is below, suitably accounting for the initial value output by the
; circuit.
; Optimization notes: disabling:
;  - EMPTY alone helps in cases: 20 -> 8, but little in time:
; 56 -> 51
;  - LEN alone helps a little in cases: 20 -> 16, a little in
; time: 56 -> 47
; both results in 16 cases (because STR-L-I fails) and 36s.
; Enabling STR-L-I2 then reduces to 4 cases and 25s.
;
```

THEOREM: correct-handrec-hear-l

$((\neg\ \text{empty}\,(\mathit{xcall})) \land \text{s-boolp}\,(\mathit{xcall}) \land (\text{len}\,(\mathit{xcall}) = \text{len}\,(\mathit{xm})))$
$\rightarrow\ \ (\text{l}\,(\text{sy-handrec}\,(\texttt{'yhear},\ \mathit{xcall},\ \mathit{xm}))$
$=\ \ \textbf{if}\ \text{empty}\,(\text{p}\,(\mathit{xcall}))\ \ \textbf{then f}$
$\textbf{else}\ \text{l}\,(\text{p}\,(\mathit{xcall}))\ \textbf{endif})$

```
; naturally we can weaken this theorem to look like Paillet:
```

THEOREM: correct-handrec-hear-paillet
$((\neg\ \text{empty}\,(\text{p}\,(\mathit{xcall}))) \land \text{s-boolp}\,(\mathit{xcall}) \land (\text{len}\,(\mathit{xcall}) = \text{len}\,(\mathit{xm})))$
$\rightarrow\ \ (\text{l}\,(\text{sy-handrec}\,(\texttt{'yhear},\ \mathit{xcall},\ \mathit{xm})) = \text{l}\,(\text{p}\,(\mathit{xcall})))$

```
; Note: we can also obtain it DIRECTLY w/ the same blastful
; induction:
; (induct (induct-P2 Xcall Xm))
; (expand (sy-HANDREC 'Yhear Xcall Xm)
;  (SY-HANDREC 'Y3 (P XCALL) (P XM))
;  (SY-HANDREC 'Y2 (P XCALL) (P XM))
;  (SY-HANDREC 'Y0 (P XCALL) (P XM))
;  (SY-HANDREC 'Y1 (P XCALL) (P XM))
;  )

; Of course, from there with a APL-split we get the string version:
; note: disabling LEN here works fine. (Never tried it enabled
; though.)
```

THEOREM: apl-split
$((\neg\ \text{empty}\,(\mathit{xcall})) \land (\text{len}\,(\mathit{xcall}) = \text{len}\,(\mathit{xm})))$
$\rightarrow\ \ (\text{sy-handrec}\,(\texttt{'yhear},\ \mathit{xcall},\ \mathit{xm})$
$=\ \ \text{a}\,(\text{p}\,(\text{sy-handrec}\,(\texttt{'yhear},\ \mathit{xcall},\ \mathit{xm})),$
$\text{l}\,(\text{sy-handrec}\,(\texttt{'yhear},\ \mathit{xcall},\ \mathit{xm}))))$

```
; Note: A-P-L is required here. The fact that it wasn't in handreco
; is probably a freak...
```

THEOREM: correct-handrec-hear-s
$(\text{s-boolp}\,(\mathit{xcall}) \land (\text{len}\,(\mathit{xcall}) = \text{len}\,(\mathit{xm})))$
$\rightarrow\ \ (\text{sy-handrec}\,(\texttt{'yhear},\ \mathit{xcall},\ \mathit{xm})$
$=\ \ \textbf{if}\ \text{empty}\,(\mathit{xcall})\ \ \textbf{then}\ \text{E}$
$\textbf{else}\ \text{i}\,(\textbf{f},\ \text{p}\,(\mathit{xcall}))\ \textbf{endif})$

```
; We can also get the string equality directly by a blastful
; induction:
```

```
; Note also that disabling LEN here is a MAJOR win: 20 cases -> 1,
; time: 31s -> 8s.
; Note also that disabling EMPTY fails the proof.  We're probably
; missing some crucial substitute for it.  Since it's not recursive
; it may not be worth it to worry about it...
```

THEOREM: correct-handrec-hear-s2
(s-boolp (*xcall*) ∧ (len (*xcall*) = len (*xm*)))
→ (sy-handrec (`'yhear`, *xcall*, *xm*)
= **if** empty (*xcall*) **then** E
else i (**f**, p (*xcall*)) **endif**)

```
; As for INFIN, the correctness property is rather trivial, except
; of course that for strings we have to correct for initial
; values...

; comb_and3.bm: Logical And combinational element, with 3 inputs
; U7-DONE
```

DEFINITION: and3 (*u1*, *u2*, *u3*) = (*u1* ∧ *u2* ∧ *u3*)

```
; Everything below generated by: (bmcomb 'and3 '() '(x1 x2 x3))
```

DEFINITION:
s-and3 (*x1*, *x2*, *x3*)
= **if** empty (*x1*) **then** E
else a (s-and3 (p (*x1*), p (*x2*), p (*x3*)), and3 (l (*x1*), l (*x2*), l (*x3*))) **endif**

```
;; A2-Begin-S-AND3
```

THEOREM: a2-empty-s-and3
empty (s-and3 (*x1*, *x2*, *x3*)) = empty (*x1*)

THEOREM: a2-e-s-and3
(s-and3 (*x1*, *x2*, *x3*) = E) = empty (*x1*)

THEOREM: a2-lp-s-and3
len (s-and3 (*x1*, *x2*, *x3*)) = len (*x1*)

THEOREM: a2-lpe-s-and3
eqlen (s-and3 (*x1*, *x2*, *x3*), *x1*)

THEOREM: a2-ic-s-and3
$((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)))$
$\rightarrow \quad (\text{s-and3}(\text{i}(c\_x1, x1), \text{i}(c\_x2, x2), \text{i}(c\_x3, x3))$
$\quad = \quad \text{i}(\text{and3}(c\_x1, c\_x2, c\_x3), \text{s-and3}(x1, x2, x3)))$

THEOREM: a2-lc-s-and3
$(\neg\ \text{empty}(x1)) \rightarrow (\text{l}(\text{s-and3}(x1, x2, x3)) = \text{and3}(\text{l}(x1), \text{l}(x2), \text{l}(x3)))$

THEOREM: a2-pc-s-and3
$\text{p}(\text{s-and3}(x1, x2, x3)) = \text{s-and3}(\text{p}(x1), \text{p}(x2), \text{p}(x3))$

THEOREM: a2-hc-s-and3
$((\neg\ \text{empty}(x1)) \land ((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3))))$
$\rightarrow \quad (\text{h}(\text{s-and3}(x1, x2, x3)) = \text{and3}(\text{h}(x1), \text{h}(x2), \text{h}(x3)))$

THEOREM: a2-bc-s-and3
$((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)))$
$\rightarrow \quad (\text{b}(\text{s-and3}(x1, x2, x3)) = \text{s-and3}(\text{b}(x1), \text{b}(x2), \text{b}(x3)))$

THEOREM: a2-bnc-s-and3
$((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)))$
$\rightarrow \quad (\text{bn}(n, \text{s-and3}(x1, x2, x3)) = \text{s-and3}(\text{bn}(n, x1), \text{bn}(n, x2), \text{bn}(n, x3)))$

```
;; A2-End-S-AND3

; eof:comb_and3.bm
 ; for spec only

; a stupid local linguistic hack, which we won't always want:
```

THEOREM: s-and3-to-s-and2
$\text{s-and3}(x1, x2, x3) = \text{s-and}(\text{s-and}(x1, x2), x3)$

EVENT: Disable s-and3-to-s-and2.

```
; Note: we may need some A2-EMPTY in here, hence no EQ-LEN hyp.,
; but we should not need any e-equalization, hence LEN disabled
```

THEOREM: correct-handrec-infin-s
$((\neg\ \text{empty}(\text{p}(\text{p}(xcall))))$
$\land \quad (\neg\ \text{empty}(\text{p}(xm)))$
$\land \quad (\text{len}(xcall) = \text{len}(xm)))$
$\rightarrow \quad (\text{sy-handrec}(\texttt{'yinfin}, xcall, xm)$
$\quad = \quad \text{i}(\mathbf{f}, \text{s-and3}(\text{p}(xcall), \text{p}(xm), \text{i}(\mathbf{t}, \text{s-not}(\text{p}(\text{p}(xcall)))))))$

```
; or in terms of last-chars, and obtained from INFIN-S:
```

THEOREM: correct-handrec-infin-l

$((\neg \text{ empty}(\text{p}(\text{p}(xcall))))$
$\land \quad (\neg \text{ empty}(\text{p}(xm)))$
$\land \quad (\text{len}(xcall) = \text{len}(xm)))$
$\rightarrow \quad (\text{l}(\text{sy-handrec}(\texttt{'yinfin}, xcall, xm))$
$\quad = \quad (\text{l}(\text{p}(xcall)) \land \text{l}(\text{p}(xm)) \land (\neg \text{ l}(\text{p}(\text{p}(xcall))))))$

```
; To prove the INITIALIZATION PROPERTY for the HANDSHAKE RECEIVER,
; Paillet #6, we take the SYSD definition for the (normal,
; initialized) sysd, and hand-MODIFY it to be EXPLICITLY
; PARAMETRIZED on the register initial values. Then we prove that
; the right inital input sequence produces the right values in
; the register (lines).
```

DEFINITION:

sy-handrec-i (*a1*, *a2*, *a3*, *ln*, *xcall*, *xm*)
= **if** *ln* = `'yrcall`
then if empty (*xcall*) **then** E
else i (*a1*, p (*xcall*)) **endif**
elseif *ln* = `'y0`
then s-not (sy-handrec-i (*a1*, *a2*, *a3*, `'yrcall`, *xcall*, *xm*))
elseif *ln* = `'y1`
then s-and (sy-handrec-i (*a1*, *a2*, *a3*, `'yrcall`, *xcall*, *xm*),
sy-handrec-i (*a1*, *a2*, *a3*, `'yhear`, *xcall*, *xm*))
elseif *ln* = `'y2`
then s-or (sy-handrec-i (*a1*, *a2*, *a3*, `'y0`, *xcall*, *xm*),
sy-handrec-i (*a1*, *a2*, *a3*, `'y1`, *xcall*, *xm*))
elseif *ln* = `'y3`
then s-and (*xcall*, sy-handrec-i (*a1*, *a2*, *a3*, `'y2`, *xcall*, *xm*))
elseif *ln* = `'y4` **then** s-and (*xcall*, *xm*)
elseif *ln* = `'y5`
then s-and (sy-handrec-i (*a1*, *a2*, *a3*, `'y4`, *xcall*, *xm*),
sy-handrec-i (*a1*, *a2*, *a3*, `'y0`, *xcall*, *xm*))
elseif *ln* = `'yhear`
then if empty (*xcall*) **then** E
else i (*a2*, sy-handrec-i (*a1*, *a2*, *a3*, `'y3`, p (*xcall*), p (*xm*))) **endif**
elseif *ln* = `'yinfin`
then if empty (*xcall*) **then** E
else i (*a3*, sy-handrec-i (*a1*, *a2*, *a3*, `'y5`, p (*xcall*), p (*xm*))) **endif**
else sfix (*xcall*) **endif**

```
; the correct initialization theorem reads:
; Note: u's are unconstrained characters
```

THEOREM: correct-handrec-init

$((\mathit{xcall\_i} = \mathrm{a}(\mathrm{a}(\mathrm{E}, \mathbf{f}), \mathit{u1})) \land (\mathit{xm\_i} = \mathrm{a}(\mathrm{a}(\mathrm{E}, \mathit{u2}), \mathit{u3})))$
$\rightarrow \quad ((\mathrm{l}(\text{sy-handrec-i}(\mathit{a1}, \mathit{a2}, \mathit{a3}, \texttt{'yrcall}, \mathit{xcall\_i}, \mathit{xm\_i})) = \mathbf{f})$
$\quad \land \quad (\mathrm{l}(\text{sy-handrec-i}(\mathit{a1}, \mathit{a2}, \mathit{a3}, \texttt{'yhear}, \mathit{xcall\_i}, \mathit{xm\_i})) = \mathbf{f})$
$\quad \land \quad (\mathrm{l}(\text{sy-handrec-i}(\mathit{a1}, \mathit{a2}, \mathit{a3}, \texttt{'yinfin}, \mathit{xcall\_i}, \mathit{xm\_i})) = \mathbf{f}))$

```
#| ))

;; THE REST OF THIS are unneeded (anymore) experiments:

(setq handrec-h1 '(
; we can try to go through the intermediate equations, like
; Paillet, using the technique we worked out in bcdS, but it's not
; clear it's worth it!

(defn sy-H1 (ln Xcall Xm)
(if (equal ln 'Yhear)
    (if (empty Xcall) (e) ; by hand
    (if (empty (P Xcall)) (A (e) F) ; by hand
        (I F (S-AND (P XCALL)
    (S-OR (S-NOT (I F (P (P XCALL))))
  (S-AND (I F (P (P XCALL)))
 (SY-H1 'YHEAR (P XCALL) (P XM))))))
))
(sfix Xcall)
))

;; NOTE that we got the expansion from BM by doing:
;;(dcl dummy ())
;;
;;(prove-lemma x1 ()
;;(implies (and (not (empty Xcall)) (not (empty (P Xcall))))
;;  (equal (sy-HANDREC 'Yhear Xcall Xm) (dummy)))
;;((expand (sy-HANDREC 'Yhear Xcall Xm)
;;  (SY-HANDREC 'Y3 (P XCALL) (P XM)) ; grabbing from proof
;;  (SY-HANDREC 'Y2 (P XCALL) (P XM)) ; as we went along..
;;  (SY-HANDREC 'Y0 (P XCALL) (P XM))
;;  (SY-HANDREC 'Y1 (P XCALL) (P XM))
;;  )
;; (do-not-induct) (do-not-generalize)
```

```
;; (disable a2-ic-s-not)
;; )
;;)

; H1 is just a GENERALIZED sysd, and our A2 lemmas should still be
; true. The following were (Sugar) generated by:
; (vp (bma2sysd-aux 'sy-H1 'sy-H1 '(x) '(and or not))) %%NOT-DONE

(prove-lemma Handrec-is-H1 (rewrite)
(equal (sy-HANDREC 'Yhear Xcall Xm)
       (sy-H1 'Yhear Xcall Xm))
((induct (induct-P2 Xcall Xm))
 (expand (sy-HANDREC 'Yhear Xcall Xm)      ; taken straight from Dummy!
 (SY-HANDREC 'Y3 (P XCALL) (P XM))
 (SY-HANDREC 'Y2 (P XCALL) (P XM))
 (SY-HANDREC 'Y0 (P XCALL) (P XM))
 (SY-HANDREC 'Y1 (P XCALL) (P XM))

 (sy-H1 'Yhear Xcall Xm)
 )
 (disable a2-ic-s-not)
 )
)

; then we could prove the same stuff as what we did for SY-HANDREC,
; but that's a rather backward way to go!


; eof: handrec.bm
))  |#
```

Index