```
#e (COND_CASES_TAC THEN REWRITE_TAC [IN_INSERT;NOT_IN_EMPTy;PAIR_EQ] THEN
#   REWRITE_TAC [RIGHT_AND_OVER_OR] THEN
#   REWRITE_TAC [LEFT_AND_OVER_OR;conj_to_cond] THEN
#   CONV_TAC (ONCE_DEPTH_CONV string_EQ_CONV) THEN
#   CONV_TAC (ONCE_DEPTH_CONV COND_CONV) THEN
#   REWRITE_TAC[]);;
OK..
goal proved
|- {(sig,val) |
    ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
    (sig,val) IN ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {})} =
   {(sig,val) |
    ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
    (sig,val) IN
    ((t >= now) => {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} | {})}
|- EQ_TAU
   (\t.
     ((t >= now + 1) =>
      {(`C`,~(a /\ b)),(`A`,a),(`B`,b)} |
      ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {})))
   (\t.
     ((t >= now + 1) =>
      {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)} |
      ((t >= now) => {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} | {})))
   {`A`,`B`,`C`}

Previous subproof:
goal proved
() : void
```

```
#let NAND_THM = GEN_ALL (top_thm());;
NAND_THM =
|- !now a b c.
      EQ_TAU (\t. (t >= now + 1) =>
                  {(`C`,~(a /\ b)),(`A`,a),(`B`,b)} |
                  ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {}))
         (\t. (t >= now + 1) =>
              {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)} |
              ((t >= now) =>
               {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} |
               {})) {`A`,`B`,`C`}
```

```
#quit();;
```

```
#e (REWRITE_TAC [EQ_TAU] THEN
#   GEN_TAC THEN BETA_TAC THEN COND_CASES_TAC THEN REWRITE_TAC [] THEN
#   CONV_TAC (ONCE_DEPTH_CONV let_CONV) THEN
#   REWRITE_TAC [IN_INSERT;NOT_IN_EMPTY;PAIR_EQ]);;
OK..
2 subgoals
"{sig,val |
 ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
 (sig,val) IN ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {})} =
    {sig,val |
    ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
    (sig,val) IN ((t >= now) =>
                  {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} |
                  {})}"
    [ "~(t >= now + 1)" ]

"{sig,val | ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
            ((sig = `C`) /\ (val = ~(a /\ b)) \/
             (sig = `A`) /\ (val = a) \/
             (sig = `B`) /\ (val = b))} =
    {sig,val | ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
               ((sig = `C`) /\ (val = ~(a /\ b)) \/
                (sig = `TMP`) /\ (val = a /\ b) \/
                (sig = `A`) /\ (val = a) \/
                (sig = `B`) /\ (val = b))}"
    [ "t >= now + 1" ]

() : void

#e (REWRITE_TAC [RIGHT_AND_OVER_OR] THEN
#   REWRITE_TAC [LEFT_AND_OVER_OR;conj_to_cond] THEN
#   CONV_TAC (ONCE_DEPTH_CONV string_EQ_CONV) THEN
#   CONV_TAC (ONCE_DEPTH_CONV COND_CONV) THEN
#   REWRITE_TAC[]);;
OK..
goal proved
|- {(sig,val) |
    ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
    ((sig = `C`) /\ (val = ~(a /\ b)) \/
     (sig = `A`) /\ (val = a) \/
     (sig = `B`) /\ (val = b))} =
   {(sig,val) |
    ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
    ((sig = `C`) /\ (val = ~(a /\ b)) \/
     (sig = `TMP`) /\ (val = a /\ b) \/
     (sig = `A`) /\ (val = a) \/
     (sig = `B`) /\ (val = b))}

Previous subproof:
"{sig,val |
 ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
 (sig,val) IN ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {})} =
    {sig,val |
    ((sig = `A`) \/ (sig = `B`) \/ (sig = `C`)) /\
    (sig,val) IN ((t >= now) =>
                  {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} |
                  {})}"
    [ "~(t >= now + 1)" ]

() : void
```

B.3 Equivalence Proof

```
#let theta' = rhs (rhs (concl (SPEC_ALL NAND_BEH)))
#and theta'' = rhs (rhs (concl (SPEC_ALL NAND_IMP)))
#and conj_to_cond = theorem `equiv-thms` `conj_to_cond`
#and EQ_TAU = definition `Femto-config` `EQ_TAU`;;
theta' =
"\t. (t >= now + 1) =>
     {(`C`,~(a /\ b)),(`A`,a),(`B`,b)} |
     ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {})"
: term
theta'' =
"\t. (t >= now + 1) =>
     {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)} |
     ((t >= now) => {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} | {})"
: term
conj_to_cond =
|- !signl A B P. (signl = A) /\ (signl = B) /\ P =
                    (A = B) => ((signl = A) /\ P) | F
EQ_TAU =
|- !tau' tau'' sigs.
      EQ_TAU tau' tau'' sigs =
         (!t. let sigma' = tau' t
              and sigma'' = tau'' t
              in ({sig,val | sig IN sigs /\ (sig,val) IN sigma'} =
                     {sig,val | sig IN sigs /\ (sig,val) IN sigma''}))

#g "EQ_TAU ^theta' ^theta'' {`A`,`B`,`C`}";;
"EQ_TAU (\t. (t >= now + 1) =>
             {(`C`,~(a /\ b)),(`A`,a),(`B`,b)} |
             ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {}))
    (\t. (t >= now + 1) =>
         {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)} |
         ((t >= now) =>
          {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} |
          {})) {`A`,`B`,`C`}"

() : void
```

```
#let th8 = BETA_RULE (RIGHT_CONV_RULE ((ONCE_DEPTH_CONV if_arith_CONV)
#                                      THENC
#                                      (DEPTH_CONV simp_ATT_CONV)) th7);;
th8 =
|- let P PROCESS (`A`,`B`)
         BEGIN
             <= ( AND ) AFTER 1 NS;
         END PROCESS;

         PROCESS (`TMP`)
         BEGIN
             <= (NOT ) AFTER 0 NS;
         END PROCESS; = PROCESS (`A`,`B`)
                        BEGIN
                            `TMP` <= (`A` AND `B`) AFTER 1 NS;
                        END PROCESS;

                        PROCESS (`TMP`)
                        BEGIN
                            `C` <= (NOT `TMP`) AFTER 0 NS;
                        END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. SND ((t >= now + 1) =>
                    ({},{(`C`,~(a/\b)),(`TMP`,(a/\b)),(`A`,a),(`B`,b)}) |
                    ((t >= now) =>
                     ({`A`},{(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))}) |
                     ({},{})))))
```

```
#let NAND_IMP = REWRITE_RULE [SND_SIMP] th8;;
NAND_IMP =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. (t >= now + 1) =>
               {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)} |
               ((t >= now) =>
                {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} |
                {})))
```

```
#let th7 = REWRITE_RULE [COND_ID;ADD_TO_THETA] th6;;
th7 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. SND ((\t. (t >= now + 1) =>
                         ({},{(`C`,~(a /\ b)),(`TMP`,(a /\ b)),
                             (`A`,a),(`B`,b)}) |
                         (\t. (now >= now + 1) =>
                              ((t >= now) =>
                               ({`A`},{(`A`,a),(`B`,b),(`C`,c),
                                       (`TMP`,~(a /\ b))}) |
                               ({},{})) |
                              ((now <= t /\ t >= now + 1) =>
                               ({`A`},{(`A`,a),(`B`,b),(`C`,c),
                                       (`TMP`,~(a /\ b))}) |
                               ({},{}))) t) t)))
```

```
#let th6 =
#   RIGHT_CONV_RULE (ONCE_DEPTH_CONV (ONCE_AROUND THENC ONCE_AROUND))
th5;;
th6 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (~(a /\ b) = c) =>
   (beh = (\t. SND (ADD_TO_THETA
                       {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),
                        (`B`,b)} {}
                       (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),
                                      (`TMP`,~(a /\ b))} {`A`}
                           (\t'. ({},{})) now (now + 1)) (now + 1)
                       (now + 1) t))) |
   (beh = (\t. SND (ADD_TO_THETA
                       {(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),
                        (`B`,b)} {}
                       (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),
                                      (`TMP`,~(a /\ b))} {`A`}
                           (\t'. ({},{})) now (now + 1)) (now + 1)
                       (now + 1) t)))
```

```
#let th5 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV finish_GAMMA) th4;;
th5 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (~(a /\ b) = c) =>
   ([0;1],(now + 1),{(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)},{},
    ADD_TO_THETA {(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)} {`TMP`}
       (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))}
           {`A`} (\t. ({},{})) now (now + 1)) (now + 1) (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
       `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,
    INERT_POST `C` (~(a /\ b)) (\t. {}) (now + 1) 0> --Sim--> beh |
   ([0;1],(now + 1),{(`C`,~(a/\b)),(`TMP`,(a/\b)),(`A`,a),(`B`,b)},{`C`},
    ADD_TO_THETA {(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)} {`TMP`}
       (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))}
           {`A`} (\t. ({},{})) now (now + 1)) (now + 1) (now + 1) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
       `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,
    INERT_POST `C` (~(a /\ b)) (\t. {}) (now + 1) 0> --Sim--> beh
```

```
#let th4 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV ONCE_AROUND) th3;;
th4 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   ([0;1],(now + 1),{(`C`,~(a /\ b)),(`TMP`,(a /\ b)),(`A`,a),(`B`,b)},
    {signl | ((signl = `C`) /\ ~(~(a /\ b) = c))},
    ADD_TO_THETA {(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)} {`TMP`}
       (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))}
          {`A`} (\t. ({},{})) now (now + 1)) (now + 1) (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
       `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,
    INERT_POST `C` (~(a /\ b)) (\t. {}) (now + 1) 0> --Sim--> beh
```

```
#let th3 = REWRITE_RULE [lemma] (SPEC "~(X/\y)" (GEN "TMP:value" th2));;
th3 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   ([0;1],(now + 1),{(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)},{`TMP`},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,~(a /\ b))} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
       `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,INERT_POST `TMP` (a /\ b) (\t. {}) now 1> --Sim--> beh
```

```
#let th2 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV finish_GAMMA) th1;;
th2 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (a /\ b = TMP) =>
   ([0;1],(now + 1),{(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)},{},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
        `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
        `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,INERT_POST `TMP` (a /\ b) (\t. {}) now 1> --Sim--> beh |
   ([0;1],(now + 1),{(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)},{`TMP`},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
       `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,INERT_POST `TMP` (a /\ b) (\t. {}) now 1> --Sim--> beh

#let lemma =
#    PROVE("!x:bool. ~(x = ~x)",
#          GEN_TAC THEN BOOL_CASES_TAC "x:value" THEN REWRITE_TAC[]);;
lemma = |- !x. ~(x = ~x)
```

```
#let expanded = ((TOP_DEPTH_CONV let_CONV) THENC
#                (ONCE_DEPTH_CONV get_delays_CS_CONV)) tm;;
expanded =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   ([0;1],now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},{`A`},
    (\t. ({},{}))) |-- <PROCESS (`A`,`B`)
                        BEGIN
                           `TMP` <= (`A` AND `B`) AFTER 1 NS;
                        END PROCESS;

                        PROCESS (`TMP`)
                        BEGIN
                           `C` <= (NOT `TMP`) AFTER 0 NS;
                        END PROCESS;,(\t. {})> --Sim--> beh

#let th1 = RIGHT_CONV_RULE ONCE_AROUND expanded;;
th1 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `TMP` <= (`A` AND `B`) AFTER 1 NS;
           END PROCESS;

           PROCESS (`TMP`)
           BEGIN
              `C` <= (NOT `TMP`) AFTER 0 NS;
           END PROCESS;
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
        {`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   ([0;1],(now + 1),{(`TMP`,(a /\ b)),(`A`,a),(`B`,b),(`C`,c)},
    {signl | ((signl = `TMP`) /\ ~(a /\ b = TMP))},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `TMP` <= (`A` AND `B`) AFTER 1 NS;
    END PROCESS;

    PROCESS (`TMP`)
    BEGIN
       `C` <= (NOT `TMP`) AFTER 0 NS;
    END PROCESS;,INERT_POST `TMP` (a /\ b) (\t. {}) now 1> --Sim--> beh
```

```
#let NAND_BEH = REWRITE_RULE [SND_SIMP] (BETA_RULE th7);;
NAND_BEH =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. (t >= now + 1) =>
               {(`C`,~(a /\ b)),(`A`,a),(`B`,b)} |
               ((t >= now) => {(`A`,a),(`B`,b),(`C`,c)} | {})))
```

B.2 Derivation for the Implementation

```
#let tm =
#   "let P =
#    PAR (PROCESS {`A`,`B`}
#                 (INERT_SIG_ASSGN `TMP` (VAND (Sig `A`) (Sig `B`)) 1))
#        (PROCESS {`TMP`}(INERT_SIG_ASSGN `C` (VNOT (Sig `TMP`)) 0)) in
#    let mu = (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
#             {`A`},\t.({},{})) in
#         Simulate mu P (\t.{}) beh";;
tm =
"let P = PROCESS (`A`,`B`)
         BEGIN
            `TMP` <= (`A` AND `B`) AFTER 1 NS;
         END PROCESS;

         PROCESS (`TMP`)
         BEGIN
            `C` <= (NOT `TMP`) AFTER 0 NS;
         END PROCESS;
 in
 let mu =
     (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c),(`TMP`,tmp)},
      {`A`},(\t. ({},{})))
 in
     mu |-- <P,(\t. {})> --Sim--> beh"
: term
```

```
#let th5 = REWRITE_RULE [ADD_TO_THETA;COND_ID] th4 ;;
th5 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. SND ((\t. (t >= now + 1) =>
                         ({},{(`C`,~(a /\ b)),(`A`,a),(`B`,b)}) |
                         (\t. (now >= now + 1) =>
                              ((t >= now) =>
                               ({`A`},{(`A`,a),(`B`,b),(`C`,c)}) |
                               ({},{})) |
                              ((now <= t /\ t >= now + 1) =>
                               ({`A`},{(`A`,a),(`B`,b),(`C`,c)}) |
                               ({},{}))) t) t)))

#let th6 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV if_arith_CONV) th5;;
th6 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. SND ((\t. (t >= now + 1) =>
                         ({},{(`C`,~(a /\ b)),(`A`,a),(`B`,b)}) |
                         (\t. (now <= t /\ t >= now + 1) =>
                              ({`A`},{(`A`,a),(`B`,b),(`C`,c)}) |
                              ({},{})) t) t)))

#let th7 = RIGHT_CONV_RULE (DEPTH_CONV simp_ATT_CONV) th6;;
th7 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (beh = (\t. SND ((\t. (t >= now + 1) =>
                         ({},{(`C`,~(a /\ b)),(`A`,a),(`B`,b)}) |
                         (\t. (t >= now) =>
                              ({`A`},{(`A`,a),(`B`,b),(`C`,c)}) |
                              ({},{})) t) t)))
```

```
#let th3 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV ONCE_AROUND) th2;;
th3 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (~(a /\ b) = c) =>
   (beh = (\t. SND (ADD_TO_THETA {(`C`,~(a /\ b)),(`A`,a),(`B`,b)}
                       {} (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)}
                              {`A`} (\t'. ({},{})) now (now + 1))
                       (now + 1) (now + 1) t))) |
   ([1],(now + 1),{(`C`,~(a /\ b)),(`A`,a),(`B`,b)},{},
    ADD_TO_THETA {(`C`,~(a /\ b)),(`A`,a),(`B`,b)} {`C`}
       (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)} {`A`} (\t. ({},{})))
                     now (now + 1)) (now + 1) (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `C` <= (`A` NAND `B`) AFTER 1 NS;
    END PROCESS;,(\t. {})> --Sim--> beh

#let th4 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV ONCE_AROUND) th3 ;;
th4 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (~(a /\ b) = c) =>
   (beh = (\t. SND (ADD_TO_THETA {(`C`,~(a /\ b)),(`A`,a),(`B`,b)}
                       {} (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)}
                              {`A`} (\t'. ({},{})) now (now + 1))
                       (now + 1) (now + 1) t))) |
   (beh = (\t. SND (ADD_TO_THETA {(`C`,~(a /\ b)),(`A`,a),(`B`,b)}
                       {} (ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)}
                              {`A`} (\t'. ({},{})) now (now + 1))
                       (now + 1) (now + 1) t)))
```

```
#let th1 = RIGHT_CONV_RULE ONCE_AROUND expanded;;
th1 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   ([1],(now + 1),{(`C`,~(a /\ b)),(`A`,a),(`B`,b)},
    {signl | ((signl = `C`) /\ ~(~(a /\ b) = c))},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `C` <= (`A` NAND `B`) AFTER 1 NS;
    END PROCESS;,INERT_POST `C` (~(a /\ b)) (\t. {}) now 1> --Sim--> beh

#let th2 = RIGHT_CONV_RULE (ONCE_DEPTH_CONV finish_GAMMA) th1;;
th2 =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   (~(a /\ b) = c) =>
   ([1],(now + 1),{(`C`,~(a /\ b)),(`A`,a),(`B`,b)},{},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `C` <= (`A` NAND `B`) AFTER 1 NS;
    END PROCESS;,INERT_POST `C` (~(a /\ b)) (\t. {}) now 1> --Sim--> beh |
   ([1],(now + 1),{(`C`,~(a /\ b)),(`A`,a),(`B`,b)},{`C`},
    ADD_TO_THETA {(`A`,a),(`B`,b),(`C`,c)} {`A`}
                 (\t. ({},{})) now (now + 1)) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `C` <= (`A` NAND `B`) AFTER 1 NS;
    END PROCESS;,INERT_POST `C` (~(a /\ b)) (\t. {}) now 1> --Sim--> beh
```

APPENDIX B

A WORKED EXAMPLE

This appendix presents a HOL session comprising the deriviation of behaviours for a NAND gate and the proof of their equivalence that was described in Chapter 6.

B.1 Derivation for the Specification

```
============================================================================
              VHDL-HOL System (Version 1.0), built on 24/2/93
============================================================================

#let tm =
#    "let P = PROCESS {`A`,`B`}
#                     (INERT_SIG_ASSGN `C`(VNAND(Sig `A`)(Sig `B`))1)
#     and mu = (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},
#               {`A`},\t.({},{})) in
#         Simulate mu P (\t.{}) beh";;
tm =
"let P = PROCESS (`A`,`B`)
         BEGIN
            `C` <= (`A` NAND `B`) AFTER 1 NS;
         END PROCESS;
 in
 let mu =
     (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
 in
     mu |-- <P,(\t. {})> --Sim--> beh"
: term

#let expanded = ((TOP_DEPTH_CONV let_CONV) THENC
#                (ONCE_DEPTH_CONV get_delays_CS_CONV)) tm;;
expanded =
|- let P = PROCESS (`A`,`B`)
           BEGIN
              `C` <= (`A` NAND `B`) AFTER 1 NS;
           END PROCESS;
   in
   let mu =
       (get_delays_CS P,now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{})))
   in
       mu |-- <P,(\t. {})> --Sim--> beh =
   ([1],now,{(`A`,a),(`B`,b),(`C`,c)},{`A`},(\t. ({},{}))) |--
   <PROCESS (`A`,`B`)
    BEGIN
       `C` <= (`A` NAND `B`) AFTER 1 NS;
    END PROCESS;,(\t. {})> --Sim--> beh
```