

User: EN3005

-at

<USERC>EN>EN3005>PROJECT. LIST


```
WWWWW W W WWW WWW WWW WWWWWW
W WW W W W W W W W W
W W W W W W W W WWW
WWW W W W WW W W W W W
W W W W W W W W W
W W WW W W W W W W W
WWWWW W W WWW WWW WWW WWW
```

```
WWW WWW WWW WWW WWWWWW WWW WWWWWW W WWW WWW WWWWWW
W W W W W W W W W W W W W W W W W W
W W W W W W W W W W W W W W W W W
WWW WWW W W W WWW W W W W W W W W
W W W W W W W W W W W W W W W W
W W W W W W W W W W W W W W W W
W W W WWW WW WWWWWW WWW W WW WWWWWW WWW WWW W
```


Label: PRT012 -form

Pathname: <USERC>EN>EN3005>PROJECT. LIST

File last modified: 86-09-25. 11: 13: 48. Thu

Spooled: 86-09-25. 12: 37: 56. Thu [Spooler rev 19. 4. 5h]

Started: 86-09-25. 12: 38: 56. Thu on: PRO by: PR3

COMPILED ON: 860925 AT: 11:12 BY: PL1G REV. 19.4.6

Options selected:

PROJECT -b PROJECT.bin -l PROJECT.list -BIG -LCASE -NESTING -FULL_OPTIMIZE

Optimization note: Currently "-OPTimize" means "-OPTimize 2",

"-Full_OPTimize" means "-OPTimize 3", and default is "-OPTimize 0".

Options used(* follows those that are not default):

64V No_Allow_PREconnection BIG* Binary COPY No_DeBuG No_ERRList ERRtty
No_EXPlist No_FRN Listing* MAP NESting* No_OFFset OPTimize(3)* No_OVerFlow
No_PRODUCTION No_Range Silent(0) TIME No_STATistics Store_Owner_Field
LCaSe* No_XRef

```

1      0
2      0
3      0      /*
4      * 0      =====
5      * 0      = UTILITY PROGRAMME FOR THE MINIMISATION OF BOOLEAN FUNCTIONS      =
6      * 0      = Written by R.C.LUCKHURST, September 1986                        =
7      * 0      = for final year BSc Electrical Engineering project, Bristol Polytechnic      =
8      * 0      =====
9      * 0      */
10     0
11     0
12     0      %REPLACE true BY '1'B, false BY '0'B;
13     0      %REPLACE low BY 1, high BY 2, cost BY 3, status BY 4;
14     0      %REPLACE redundant BY 1, min_cost_redundant BY 2, non_essential BY 3, min_cost_essential BY 4, essential BY 5;
15     0
16     0      BSc_project: PROCEDURE OPTIONS (MAIN);
17     1
18     1      DECLARE (BINARY, BIT, CEIL, CHARACTER, COPY, DECIMAL, INDEX,
19     1              LENGTH, LOG2, MIN, SUBSTR, TRANSLATE, VERIFY) BUILTIN,
20     1
21     1              (num_minterms, num_dont_cares, num_terms, num_pis, num_ne_pis, num_inepi_sums,
22     1              minterm (512), dont_care (512), term (1024), p_i (4,256), ne_pi (96),
23     1              function_order, solution_cost) FIXED BINARY,
24     1
25     1              (unique_solution, new_data,
26     1              epi_covers_minterm (256), pi_covers_minterm (256,256)) BIT STATIC,
27     1              ine_pi_sum (3000) BIT (96) ALIGNED,
28     1
29     1              version CHARACTER (4) STATIC INITIAL ('V1.0'),
30     1              continue CHARACTER (30) VARYING,
31     1              pi_status (5) CHARACTER(30) VARYING STATIC INITIAL
32     1              ('redundant', 'minimum-cost redundant', 'non-essential', 'minimum-cost essential', 'essential'),
33     1              results_file FILE;
34     1
35     1
36     1
37     1      /*
38     * 1      #####
39     * 1      # UTILITY PROCEDURES #
40     * 1      #####
41     * 1      */
42     1
43     1
44     1      /*****
45     * 1      * PROCEDURE equivalent: Returns logical equivalence between 2 integers      *
46     * 1      *****/
47     1      equivalent: PROCEDURE (x,y) RETURNS (FIXED BINARY);
48     2          DECLARE (x,y) FIXED BINARY;
49     2          RETURN (x & y ! ^ x & ^ y);

```



```

110      4      END;
111      3      END;
112      2
113      2      /* now extract sorted terms back into ordered minterm & dont-care arrays */
114      2      num_minterms = 0; num_dont_cares = 0;
115      2      DO t = 1 TO num_terms;
116      3          IF type(t) THEN DO;
117      4              num_minterms = num_minterms + 1;
118      4              minterm(num_minterms) = term(t);
119      4              END;
120      3          ELSE DO;
121      4              num_dont_cares = num_dont_cares + 1;
122      4              dont_care(num_dont_cares) = term(t);
123      4              END;
124      3      END;
125      2      END sort_data;
126      1
127      1
128      1      /*
129      * 1      #####
130      * 1      # INPUT PROCEDURES #
131      * 1      #####
132      * 1      */
133      1
134      1
135      1      /*****
136      * 1      * PROCEDURE menu_selection: Returns menu item requested: 1 - 5 *
137      * 1      *****/
138      1      menu_selection: PROCEDURE RETURNS (FIXED BINARY);
139      2          DECLARE menu_item CHARACTER (30) VARYING, m FIXED BINARY;
140      2          DO WHILE (true);
141      3              GET LIST (menu_item);
142      3              IF VERIFY(menu_item, '12345') = 0 THEN DO;
143      4                  m = BINARY(menu_item);
144      4                  IF m >= 1 & m <= 5 THEN RETURN (m);
145      4                  END;
146      3          END;
147      2      END menu_selection;
148      1
149      1
150      1      /*****
151      * 1      * PROCEDURE continue_prompt: Stops screen scrolling *
152      * 1      *****/
153      1      continue_prompt: PROCEDURE;
154      2          PUT SKIP(2) LIST ('Press RETURN to continue -->');
155      2          GET LIST (continue);
156      2      END continue_prompt;
157      1
158      1
159      1      /*****
160      * 1      * PROCEDURE enter_data: Used to enter minterms and don't cares *
161      * 1      *****/
162      1      enter_data: PROCEDURE;
163      2          DECLARE action CHARACTER (30) VARYING, deleted BIT ALIGNED, (i, t) FIXED BINARY;
164      2
165      2          get_input_list: PROCEDURE;
166      3              DECLARE (upper, lower) FIXED BINARY,
167      3                  input_item CHARACTER (30) VARYING,
168      3                  illegal_entry BIT ALIGNED;
169      3

```

```

170      num_terms = 0;
171      illegal_entry = false;
172      PUT EDIT ('Enter values in the range 0 to 255 seperated by commas or blanks or returns. ',
173              'A range of values may be entered using a hyphen, e.g. 10-15. ',
174              'Type E after the last entry. ',
175              '--> ') (SKIP,A);
176      DO WHILE (true);
177          GET LIST (input_item);
178          IF TRANSLATE(input_item, 'E', 'e') = 'E' THEN RETURN;
179          ELSE IF VERIFY(input_item, '-0123456789') = 0 & INDEX(input_item, '-') ^= 1 THEN DO;
180              t = INDEX(input_item, '-');
181              IF t = 0 THEN DO;
182                  IF num_terms <= 255 THEN DO;
183                      num_terms = num_terms + 1;
184                      term(num_terms) = BINARY(input_item);
185                      END;
186                  ELSE;
187                      END;
188                  ELSE IF t <= 5 & LENGTH(input_item) - t < 5 THEN DO;
189                      upper = BINARY(SUBSTR(input_item, t + 1)); IF upper > 255 THEN upper = 255;
190                      lower = BINARY(SUBSTR(input_item, 1, t - 1)); IF lower > 255 THEN lower = 255;
191                      IF lower > upper THEN DO; t = lower; lower = upper; upper = t; END;
192                      DO t = lower TO upper WHILE (num_terms <= 255);
193                          num_terms = num_terms + 1;
194                          term(num_terms) = t;
195                          END;
196                      END;
197                  ELSE illegal_entry = true;
198                  END;
199              IF illegal_entry THEN PUT SKIP LIST ('Illegal entries have been disregarded. ');
200      END get_input_list;
201
202      DO WHILE (true);
203      PUT SKIP(3);
204      CALL sort_data;
205      CALL print_header_message(SYSPRINT);
206      CALL print_input_data(SYSPRINT);
207      PUT SKIP(3) EDIT ('C = Clear data',
208                      'AM = Add Minterms',
209                      'DM = Delete Minterms',
210                      'AD = Add Don''t cares',
211                      'DD = Delete Don''t cares',
212                      'E = End data entry',
213                      'Enter C/AM/DM/AD/DD/E --> ') (6(COLUMN(24), A, SKIP), SKIP, COLUMN(24), A);
214      action = ' ';
215      DO WHILE (VERIFY(action, 'ACDEM') ^= 0);
216          GET LIST (action);
217          action = TRANSLATE(action, 'ACDEM', 'acdem');
218          END;
219      IF action = 'E' THEN RETURN;
220      IF action = 'C' THEN DO;
221          num_minterms = 0; num_dont_cares = 0;
222          END;
223      IF action = 'AM' | action = 'DM' | action = 'AD' | action = 'DD' THEN DO;
224          CALL get_input_list;
225          IF action = 'AM' THEN DO; /* add input list to minterms */
226              DO t = 1 TO num_terms;
227                  minterm(num_minterms + t) = term(t);
228                  END;
229              num_minterms = num_minterms + num_terms;

```

```

230      5      END;
231      4      ELSE IF action = 'DM' THEN DO; /* make minterms contained in i/p list out of range for deletion w
232      5      DO t = 1 TO num_terms;
233      6      deleted = false;
234      6      DO i = 1 TO num_minterms WHILE (^ deleted);
235      7      IF minterm(i) = term(t) THEN DO;
236      8      minterm(i) = -1;
237      8      deleted = true;
238      8      END;
239      7      END;
240      6      END;
241      5      END;
242      4      ELSE IF action = 'AD' THEN DO; /* add input list to dont-cares */
243      5      DO t = 1 TO num_terms;
244      6      dont_care(num_dont-cares + t) = term(t);
245      6      END;
246      5      num_dont-cares = num_dont-cares + num_terms;
247      5      END;
248      4      ELSE DO; /* make dont-cares contained in i/p list out of range for deletion when sorted */
249      5      DO t = 1 TO num_terms;
250      6      deleted = false;
251      6      DO i = 1 TO num_dont-cares WHILE (^ deleted);
252      7      IF dont_care(i) = term(t) THEN DO;
253      8      dont_care(i) = -1;
254      8      deleted = true;
255      8      END;
256      7      END;
257      6      END;
258      5      END;
259      4      IF num_terms > 0 THEN new_data = true;
260      4      END;
261      3      END;
262      2      END enter_data;
263      1
264      1
265      1      /*
266      * 1      #####
267      * 1      # MINIMISATION PROCEDURES #
268      * 1      #####
269      * 1      */
270      1
271      1
272      1      /*#####
273      * 1      * PROCEDURE prime_implicants: Generates complete list of PIs *
274      * 1      #####*/
275      1      prime_implicants: PROCEDURE;
276      2
277      2      DECLARE (i, j, term_i, term_j, i_eqv_j, vertex, p, m) FIXED BINARY,
278      2      (all_vertices_contained, covered) BIT ALIGNED;
279      2
280      2      /* generate the prime implicants */
281      2      num_pis = 0;
282      2      DO i = 1 TO num_terms;
283      3      DO j = num_terms TO i BY -1;
284      4      /* choose the pair (i, j) */
285      4      term_i = term(i); term_j = term(j);
286      4      /* is (i, j) a cell? */
287      4      IF (term_i & term_j) = term_i THEN DO;
288      5      /* are all the vertices of (i, j) in the function? */
289      5      i_eqv_j = equivalent(term_i, term_j);

```

```

290      5      all_vertices_contained = true; m = i + 1;
291      5      DO vertex = term_i + 1 TO term_j - 1 WHILE (all_vertices_contained);
292      6          IF (i_eqv_j & vertex) = term_i THEN DO;
293      7              DO WHILE (term(m) < vertex); m = m + 1; END;
294      7              all_vertices_contained = (term(m) = vertex);
295      7              m = m + 1;
296      7              END;
297      6          END;
298      5      IF all_vertices_contained THEN DO;
299      6          /* is (i,j) covered by an entry in the p.i. table? */
300      6          covered = false;
301      6          IF num_pis ^= 0 THEN DO p = 1 TO num_pis WHILE (^ covered);
302      7              IF term_j <= p_i(high,p) THEN
303      7                  IF (p_i(low,p) & term_i) = p_i(low,p) THEN
304      7                      covered = ((term_j & p_i(high,p)) = term_j);
305      7                  END;
306      6              IF ^ covered THEN DO;
307      7                  num_pis = num_pis + 1;
308      7                  p_i(low,num_pis) = term_i;
309      7                  p_i(high,num_pis) = term_j;
310      7                  END;
311      6              END;
312      5          END;
313      4      END;
314      3      END;
315      2
316      2      END prime_implicants;
317      1
318      1
319      1      /*****
320      * 1      * PROCEDURE p_i_chart: Makes a PI chart as a bit array      *
321      * 1      *****/
322      1      p_i_chart: PROCEDURE;
323      2          DECLARE (m, p) FIXED BINARY;
324      2          /* generate the prime implicant chart */
325      2          DO m = 1 TO num_minterms;
326      3              DO p = 1 TO num_pis;
327      4                  pi_covers_minterm(p,m) = ((p_i(low,p) & minterm(m)) = p_i(low,p)
328      4                      & (minterm(m) & p_i(high,p)) = minterm(m));
329      4              END;
330      3          END;
331      2      END p_i_chart;
332      1
333      1
334      1      /*****
335      * 1      * PROCEDURE p_i_status: Categorises PIs as essential/nonessential/redundant *
336      * 1      *****/
337      1      p_i_status: PROCEDURE;
338      2          DECLARE (m, p, epi, num_covers) FIXED BINARY;
339      2
340      2          /* initialise all p.i. status to redundant */
341      2          DO p = 1 TO num_pis; p_i(status,p) = redundant; END;
342      2          /* find essential p.i.s */
343      2          DO m = 1 TO num_minterms;
344      3              num_covers = 0;
345      3              DO p = 1 TO num_pis;
346      4                  IF pi_covers_minterm(p,m) THEN DO;
347      5                      epi = p;
348      5                      num_covers = num_covers + 1;
349      5                  END;

```

```

350      4      END;
351      3      IF num_covers = 1 THEN p_i(status, epi) = essential;
352      3      END;
353      2      /* find minterms covered by essential p.i.s */
354      2      num_covers = 0;
355      2      DO m = 1 TO num_minterms;
356      3          epi_covers_minterm(m) = false;
357      3          DO p = 1 TO num_pis WHILE (^ epi_covers_minterm(m));
358      4              IF p_i(status, p) = essential & pi_covers_minterm(p, m) THEN DO;
359      5                  epi_covers_minterm(m) = true;
360      5                  num_covers = num_covers + 1;
361      5              END;
362      4          END;
363      3      END;
364      2      /* determine whether 1 solution or more */
365      2      unique_solution = (num_covers = num_minterms);
366      2      /* find non-essential p.i.s */
367      2      IF ^ unique_solution THEN DO;
368      3          DO m = 1 TO num_minterms;
369      4              IF ^ epi_covers_minterm(m) THEN DO p = 1 TO num_pis;
370      5                  IF p_i(status, p) = redundant & pi_covers_minterm(p, m) THEN p_i(status, p) = non_essential;
371      5              END;
372      4          END;
373      3          /* make a table of n.e.p.i. pointers */
374      3          num_ne_pis = 0;
375      3          DO p = 1 TO num_pis;
376      4              IF p_i(status, p) = non_essential THEN DO;
377      5                  num_ne_pis = num_ne_pis + 1; ne_pi(num_ne_pis) = p;
378      5              END;
379      4          END;
380      3      END;
381      2      END p_i_status;
382      2
383      1
384      1
385      1      /*****
386      * 1      * PROCEDURE p_i_cost: Calculates literal costs of PIs *
387      * 1      *****/
388      1      p_i_cost: PROCEDURE;
389      2          DECLARE (p, l, b, literals) FIXED BINARY;
390      2          DO p = 1 TO num_pis;
391      3              p_i(cost, p) = 0; b = 1;
392      3              literals = equivalent(p_i(low, p), p_i(high, p));
393      3              DO l = 1 TO function_order;
394      4                  IF (b & literals) ^= 0 THEN p_i(cost, p) = p_i(cost, p) + 1;
395      4                  b = b + b;
396      4              END;
397      3          END;
398      2      END p_i_cost;
399      1
400      1
401      1      /*****
402      * 1      * PROCEDURE irredundand_nepi_sums: Performs algebraic conversion of *
403      * 1      * nonessential PI product-of-sums to *
404      * 1      * sum-of-products *
405      * 1      *****/
406      1      irredundant_nepi_sums: PROCEDURE;
407      2          DECLARE (m, p, c, s, num_umin_nepis) FIXED BINARY,
408      2          (b, umin_nepis(256)) BIT (96) ALIGNED,
409      2          redundant_sums BIT ALIGNED;

```



```

410      2
411      2      /* make an array of bit strings holding non-ess p.i. coverage of uncovered minterms */
412      2      num_umin_nepis = 0;
413      2      DO m = 1 TO num_minterms;
414      3          IF ^ epi_covers_minterm(m) THEN DO;
415      4              num_umin_nepis = num_umin_nepis + 1;
416      4              umin_nepis(num_umin_nepis) = 0;
417      4              b = BIT(0,95) !! '1'B;
418      4              DO p = 1 TO num_ne_pis;
419      5                  IF pi_covers_minterm(ne_pi(p),m) THEN
420      5                      umin_nepis(num_umin_nepis) = umin_nepis(num_umin_nepis) ! b;
421      5                      b = SUBSTR(b,2);
422      5                  END;
423      4              END;
424      3          END;
425      2
426      2      /* first pass - i.n.e.p.i. sums are those covering 1st uncovered minterm */
427      2      num_inepi_sums = 0;  b = BIT(0,95) !! '1'B;
428      2      DO p = 1 TO num_ne_pis;
429      3          /* if 1st uncovered minterm is covered by this n.e.p.i. then ... */
430      3          IF (umin_nepis(1) & b) ^= BIT(0,96) THEN DO;
431      4              /* ... this sum is initially this n.e.p.i. */
432      4              num_inepi_sums = num_inepi_sums + 1;  ineppi_sum(num_inepi_sums) = b;
433      4              END;
434      3              b = SUBSTR(b,2);
435      3          END;
436      2
437      2      /* continue by repeatedly combining with n.e.p.i. terms of succeeding minterms algebraically */
438      2      DO m = 2 TO num_umin_nepis;
439      3          /* initialise cover counter and n.e.p.i. pointer */
440      3          c = -1;  b = BIT(0,95) !! '1'B;
441      3          /* add each n.e.p.i. covering this minterm successively to each sum */
442      3          DO p = 1 TO num_ne_pis;
443      4              /* if this n.e.p.i. covers this minterm then ... */
444      4              IF (umin_nepis(m) & b) ^= BIT(0,96) THEN DO;
445      5                  /* ... increment cover counter for this minterm ... */
446      5                  c = c + 1;
447      5                  /* ... step through the sums for this cover ... */
448      5                  DO s = c * num_inepi_sums + 1 TO (c + 1) * num_inepi_sums;
449      6                      /* ... make a copy of current sums for next cover ... */
450      6                      ineppi_sum(s + num_inepi_sums) = ineppi_sum(s);
451      6                      /* ... add this cover to the sum */
452      6                      ineppi_sum(s) = ineppi_sum(s) ! b;
453      6                  END;
454      5              END;
455      4              b = SUBSTR(b,2);
456      4          END;
457      3          /* calculate the new number of sums resulting from above */
458      3          num_inepi_sums = (c + 1) * num_inepi_sums;
459      3          /* some sums may cover others so minimise by nulling redundant sums */
460      3          redundant_sums = false;
461      3          DO s = 1 TO num_inepi_sums;
462      4              IF ineppi_sum(s) ^= BIT(0,96) THEN DO c = 1 TO num_inepi_sums;
463      5                  IF c ^= s & ineppi_sum(c) ^= BIT(0,96) & (ineppi_sum(s) & ineppi_sum(c)) = ineppi_sum(s) THEN DO;
464      6                      ineppi_sum(c) = BIT(0,96);
465      6                      redundant_sums = true;
466      6                  END;
467      5              END;
468      4          END;
469      3          /* remove redundant sums and calculate the new number of sums resulting */

```

```

470      3      IF redundant_sums THEN CALL remove_redundant_sums;
471      3      END;
472      2
473      2      END irredundant_nepi_sums;
474      1
475      1
476      1      /*****
477      * 1      * PROCEDURE minimum_cost_solution: Finds set of minimum literal cost      *
478      * 1      *      nonessential PI sums      *
479      * 1      *****/
480      1      minimum_cost_solution: PROCEDURE;
481      2      DECLARE (s, min_cost, sum_cost(1000)) FIXED BINARY,
482      2      redundant_sums BIT ALIGNED;
483      2
484      2      /* make a table of irredundant n.e.p.i. literal costs */
485      2      DO s = 1 TO num_inepi_sums;
486      3      sum_cost(s) = nonessential_cost(s);
487      3      END;
488      2      /* find the minimum cost */
489      2      min_cost = sum_cost(1);
490      2      DO s = 2 TO num_inepi_sums;
491      3      IF sum_cost(s) < min_cost THEN min_cost = sum_cost(s);
492      3      END;
493      2      /* remove all but minimum cost sums */
494      2      DO s = 1 TO num_inepi_sums;
495      3      IF sum_cost(s) > min_cost THEN DO;
496      4      ineipi_sum(s) = BIT(0,96);
497      4      redundant_sums = true;
498      4      END;
499      3      END;
500      2      IF redundant_sums THEN CALL remove_redundant_sums;
501      2
502      2      END minimum_cost_solution;
503      1
504      1
505      1      /*****
506      * 1      * PROCEDURE ammend_p_i_status: Recategorises some nonessential PIs as      *
507      * 1      *      minimum cost essential/minimum cost redundant *
508      * 1      *****/
509      1      ammend_p_i_status: PROCEDURE;
510      2      DECLARE (ess_pis, red_pis, b) BIT (96) ALIGNED,
511      2      s FIXED BINARY;
512      2
513      2      /* find n.e.p.i.s common to each sum and those which have been removed */
514      2      ess_pis = ineipi_sum(1);
515      2      red_pis = ^ ess_pis;
516      2      DO s = 2 TO num_inepi_sums;
517      3      ess_pis = ess_pis & ineipi_sum(s);
518      3      red_pis = red_pis & ^ ineipi_sum(s);
519      3      END;
520      2      /* remove common n.e.p.i.s from the sums - these are minimum-cost essential */
521      2      DO s = 1 TO num_inepi_sums;
522      3      ineipi_sum(s) = ineipi_sum(s) & ^ ess_pis;
523      3      END;
524      2      /* ammend p.i. status table to show minimum-cost essential/redundant p.i.s */
525      2      b = BIT(0,95) || '1'B;
526      2      DO s = 1 TO num_ne_pis;
527      3      IF (ess_pis & b) ^= BIT(0,96) THEN p_i(status,ne_pi(s)) = min_cost_essential;
528      3      ELSE IF (red_pis & b) ^= BIT(0,96) THEN p_i(status,ne_pi(s)) = min_cost_redundant;
529      3      b = SUBSTR(b,2);

```

```

530      3      END;
531      2
532      2      END ammend_p_i_status;
533      1
534      1
535      1      /*****
536      * 1      * PROCEDURE remove_redundant_sums: Cleans up irredundant nonessential PI      *
537      * 1      *      sum-of-products array      *
538      * 1      *****/
539      1      remove_redundant_sums: PROCEDURE;
540      2      DECLARE (i, j) FIXED BINARY, sum_moved BIT ALIGNED;
541      2
542      2      DO i = 1 TO num_inepi_sums;
543      3      IF inepi_sum(i) = BIT(0,96) THEN DO;
544      4      sum_moved = false;
545      4      DO j = i + 1 TO num_inepi_sums WHILE (^ sum_moved);
546      5      IF inepi_sum(j) ^= BIT(0,96) THEN DO;
547      6      inepi_sum(i) = inepi_sum(j); inepi_sum(j) = BIT(0,96);
548      6      sum_moved = true;
549      6      END;
550      5      END;
551      4      IF ^ sum_moved THEN DO;
552      5      num_inepi_sums = i - 1;
553      5      RETURN;
554      5      END;
555      4      END;
556      3      END;
557      2
558      2      END remove_redundant_sums;
559      1
560      1
561      1      /*****
562      * 1      * PROCEDURE essential_cost: Returns literal cost of all essential PIs      *
563      * 1      *****/
564      1      essential_cost: PROCEDURE RETURNS (FIXED BINARY);
565      2      DECLARE (p, e_cost) FIXED BINARY;
566      2      e_cost = 0;
567      2      DO p = 1 TO num_pis;
568      3      IF p_i(status,p) > non_essential THEN e_cost = e_cost + p_i(cost,p);
569      3      END;
570      2      RETURN (e_cost);
571      2      END essential_cost;
572      1
573      1
574      1      /*****
575      * 1      * PROCEDURE nonessential_cost: Returns literal cost of all nonessential PIs      *
576      * 1      *      in specified sum-of-product sum      *
577      * 1      *****/
578      1      nonessential_cost: PROCEDURE (s) RETURNS (FIXED BINARY);
579      2      DECLARE (s, p, ne_cost) FIXED BINARY,
580      2      b BIT (96) ALIGNED;
581      2      ne_cost = 0; b = BIT(0,95) || '1'B;
582      2      DO p = 1 TO num_ne_pis;
583      3      IF (inepi_sum(s) & b) ^= BIT(0,96) THEN ne_cost = ne_cost + p_i(cost,ne_pi(p));
584      3      b = SUBSTR(b,2);
585      3      END;
586      2      RETURN (ne_cost);
587      2      END nonessential_cost;
588      1
589      1

```

```

590      1      /*****
591      *      1      * PROCEDURE run_minimisation: Performs minimisation of switching function *
592      *      1      *****/
593      1      run_minimisation: PROCEDURE;
594      2          function_order = LOG2(term(num_terms)) + 1;
595      2          PUT SKIP LIST ('(finding prime implicants)');
596      2          CALL prime_implicants; /* generates complete set of prime implicants */
597      2          CALL p_i_chart; /* generates array of pi coverage of minterms */
598      2          CALL p_i_status; /* gives ess/noness/red status to p.i.s & e.p.i. cover status to minterms & decides if uni
599      2          CALL p_i_cost; /* finds literal costs of p.i.s */
600      2          IF ^ unique_solution THEN DO;
601      3              PUT SKIP LIST ('(finding minimum cost solution)');
602      3              CALL irredundant_nepi_sums; /* generate irredundant n.e.p.i. sums to cover remaining minterms */
603      3              CALL minimum_cost_solution; /* finds lowest literal cost solutions from irredundant n.e.p.i. sums */
604      3              CALL ammend_p_i_status; /* gives min-cost-ess/min-cost-red status to n.e.p.i.s */
605      3              END;
606      2          solution_cost = essential_cost();
607      2          IF ^ unique_solution THEN solution_cost = solution_cost + nonessential_cost(1);
608      2      END run_minimisation;
609      1
610      1
611      1      /*
612      *      1      #####
613      *      1      #                                OUTPUT PROCEDURES                                #
614      *      1      #####
615      *      1      */
616      1
617      1
618      1      /*****
619      *      1      * PROCEDURE print_header_message: Prints title and version no to screen/file *
620      *      1      *****/
621      1      print_header_message: PROCEDURE (f);
622      2          DECLARE f FILE VARIABLE;
623      2          PUT FILE (f) EDIT ('BOOLEAN MINIMISATION ', version, COPY('=',26))
624      2              (COLUMN(20), A, A, SKIP, COLUMN(20), A);
625      2      END print_header_message;
626      1
627      1
628      1      /*****
629      *      1      * PROCEDURE print_menu: Prints programme menu *
630      *      1      *****/
631      1      print_menu: PROCEDURE;
632      2          CALL print_header_message(SYSPRINT);
633      2          PUT SKIP(3) LIST (' A utility for the logical minimisation of boolean functions. ');
634      2          PUT SKIP(4) EDIT ('Menu',
635      2              '-----',
636      2              '1. Enter data',
637      2              '2. Minimise',
638      2              '3. File results',
639      2              '4. Information',
640      2              '5. Quit',
641      2              'Enter 1-5 --> ');
642      2              (COLUMN(28), A, SKIP, COLUMN(28), A, SKIP(2), 5(COLUMN(24), A, SKIP), SKIP, COLUMN(24), A);
643      2      END print_menu;
644      1
645      1
646      1      /*****
647      *      1      * PROCEDURE print_input_data: Prints minterms and don't cares to screen/file *
648      *      1      *****/
649      1      print_input_data: PROCEDURE (f);

```

```

650      2      DECLARE f FILE VARIABLE, t FIXED BINARY;
651      2      /* list the minterms */
652      2      PUT FILE(f) SKIP(3) LIST ('Minterms:');
653      2      IF num_minterms = 0 THEN PUT FILE(f) SKIP LIST ('*** none ***');
654      2      ELSE PUT FILE(f) SKIP EDIT ((trim(minterm(t)) DO t = 1 TO num_minterms)) (A,X(1));
655      2      /* list the dont-cares */
656      2      PUT FILE(f) SKIP(3) LIST ('Don't cares:');
657      2      IF num_dont_cares = 0 THEN PUT FILE(f) SKIP LIST ('*** none ***');
658      2      ELSE PUT FILE(f) SKIP EDIT ((trim(dont_care(t)) DO t = 1 TO num_dont_cares)) (A,X(1));
659      2      END print_input_data;
660      1
661      1
662      1      /*****
663      * 1      * PROCEDURE output_results: Prints minimisation results to screen/file      *
664      * 1      *****/
665      1      output_results: PROCEDURE (f);
666      2      DECLARE f FILE VARIABLE;
667      2
668      2      /* print header message */
669      2      CALL print_header_message(f);
670      2
671      2      /*list minterms and dont-cares */
672      2      CALL print_input_data(f);
673      2
674      2      /* print the function order */
675      2      PUT FILE(f) SKIP(3) EDIT ('The function order is ',trim(function_order)) (A);
676      2
677      2      /* list the prime implicants and associated qualities */
678      2      cell: PROCEDURE (p) RETURNS (CHARACTER (10) VARYING);
679      3      DECLARE p FIXED BINARY;
680      3      RETURN (trim(p_i(low,p)) || ', ' || trim(p_i(high,p)));
681      3      END cell;
682      2      literals: PROCEDURE (p) RETURNS (CHARACTER (10) VARYING);
683      3      DECLARE (p, l, b) FIXED BINARY, lits CHARACTER (10) VARYING;
684      3      lits = ''; b = 1;
685      3      DO l = 1 TO function_order;
686      4      IF (equivalent(p_i(low,p),p_i(high,p)) & b) ^= 0 THEN
687      4      IF (p_i(low,p) & b) ^= 0 THEN lits = '1' || lits;
688      4      ELSE lits = '0' || lits;
689      4      ELSE lits = '-' || lits;
690      4      b = b + b;
691      4      END;
692      3      RETURN (lits);
693      3      END literals;
694      2      BEGIN;
695      3      DECLARE p FIXED BINARY;
696      3      PUT FILE(f) SKIP(3) LIST ('Prime Implicants:');
697      3      PUT FILE(f) SKIP EDIT (' p.i. ', 'cell', 'literals', 'cost', 'status')
698      3      (A,COLUMN(11),A,COLUMN(25),A,COLUMN(40),A,COLUMN(50),A);
699      3      DO p = 1 TO num_pis;
700      4      PUT FILE(f) SKIP EDIT (p,cell(p),literals(p),p_i(cost,p),pi_status(p_i(status,p)))
701      4      (F(4),COLUMN(11),A,COLUMN(25),A,COLUMN(40),F(3),COLUMN(50),A);
702      4      END;
703      3      END;
704      2
705      2      /* abort if no minterms */
706      2      IF num_minterms = 0 THEN RETURN;
707      2
708      2      /* print the prime implicant chart */
709      2      tick: PROCEDURE (b) RETURNS (CHARACTER);

```

```

710      3      DECLARE b BIT;
711      3      IF b THEN RETURN ('*');
712      3      RETURN (' ');
713      3      END tick;
714      2      BEGIN;
715      3      DECLARE (m, b, p, num_blocks, mins_per_block) FIXED BINARY;
716      3      PUT FILE(f) SKIP(3) LIST ('Prime Implicant Chart:');
717      3      num_blocks = CEIL(DECIMAL(num_minterms) / 19.0);
718      3      mins_per_block = CEIL(DECIMAL(num_minterms) / DECIMAL(num_blocks));
719      3      DO b = 1 TO num_blocks;
720      4          PUT FILE(f) SKIP LIST ('      minterm -->');
721      4          PUT FILE(f) SKIP EDIT
722      4              (' p.i. ', (minterm(m)
723      4                  DO m = (b - 1) * mins_per_block + 1 TO MIN(num_minterms, b * mins_per_block))) (A, F(3), 18(F(4)
724      4          DO p = 1 TO num_pis;
725      5              IF p_i(status, p) > redundant THEN PUT FILE(f) SKIP EDIT
726      5                  (p, (tick(pi_covers_minterm(p, m))
727      5                      DO m = (b - 1) * mins_per_block + 1 TO MIN(num_minterms, b * mins_per_block))) (F(4), 19(X
728      5              END;
729      4          END;
730      3      END;
731      2
732      2      /* print the solution */
733      2      BEGIN;
734      3      DECLARE (p, s) FIXED BINARY, or CHARACTER (2) VARYING, all_covered BIT ALIGNED, b BIT (96) ALIGNED;
735      3      IF unique_solution THEN PUT FILE(f) SKIP(3) LIST ('Unique Solution:');
736      3      ELSE PUT FILE(f) SKIP(3) LIST ('Minimum Cost Solution:');
737      3      PUT FILE(f) SKIP EDIT (' F = ') (A);
738      3      /* essentials first */
739      3      or = '';
740      3      DO p = 1 TO num_pis;
741      4          IF p_i(status, p) > non_essential THEN DO;
742      5              PUT FILE(f) EDIT (or, trim(p)) (A);
743      5              or = '+';
744      5          END;
745      4      END;
746      3      /* if these do not cover all minterms then ... */
747      3      all_covered = true;
748      3      IF ^ unique_solution THEN DO s = 1 TO num_minterms WHILE (all_covered);
749      4          all_covered = false;
750      4          DO p = 1 TO num_pis WHILE (^ all_covered);
751      5              IF p_i(status, p) > non_essential THEN
752      5                  all_covered = pi_covers_minterm(p, s);
753      5              END;
754      4          END;
755      3      /* ... minimum cost nonessentials */
756      3      IF ^ all_covered THEN DO;
757      4          PUT FILE(f) EDIT (or) (A);
758      4          DO s = 1 TO num_inepi_sums;
759      5              PUT FILE(f) EDIT (('(') (A);
760      5              or = ''; b = BIT(0, 95) !! '1'B;
761      5              DO p = 1 TO num_ne_pis;
762      6                  IF (inepi_sum(s) & b) ^= BIT(0, 96) THEN DO;
763      7                      PUT FILE(f) EDIT (or, trim(ne_pi(p))) (A);
764      7                      or = '+';
765      7                  END;
766      6                  b = SUBSTR(b, 2);
767      6              END;
768      5          PUT FILE(f) EDIT (')') (A);
769      5      END;

```

```

770      4      PUT FILE(f) SKIP LIST ('(parenthesised expressions are alternatives)');
771      4      END;
772      3      END;
773      2
774      2      /* print the literal cost of the solution */
775      2      PUT FILE(f) SKIP(3) EDIT ('Cost = ',trim(solution_cost), ' literals') (A);
776      2
777      2      END output_results;
778      1
779      1
780      1      /*****
781      * 1      * PROCEDURE print information: Prints information about the programme      *
782      * 1      *****/
783      1      print_information: PROCEDURE;
784      2      PUT SKIP(2) EDIT (
785      2      ' This utility determines the minimal 2-level solution for a Boolean switching',
786      2      'function. This function must be fully specified as minterm and don''t care',
787      2      'arrays. The maximum number of input variables is 8. Data is entered as decimal',
788      2      'values in the range 0 to 255 in any order. Ranges of values may be entered by',
789      2      'using a hyphen, e.g. 10-15. This data is sorted by the programme. If any value',
790      2      'is specified as both a minterm and a don''t care term then it is assumed to be',
791      2      'a minterm. Values which are out of range are ignored.',
792      2      ' Minimisation is done by first finding the prime implicants of the function',
793      2      'and then reducing the PI chart. Prime implicants are found by taking pairs of',
794      2      'terms (minterms or don''t cares) and testing to see if they form a cell. If they',
795      2      'do then a search is made to determine whether all the vertices of the cell are',
796      2      'either minterms or don''t cares. If so the cell is tested for containment by',
797      2      'any PI already found. If it is not contained then this cell is a PI. PI chart',
798      2      'reduction is done using the algebraic method after removing essential PIs and',
799      2      'the minterms they cover') (SKIP,A);
800      2      END print_information;
801      1
802      1
803      1      /*
804      * 1      *****/
805      * 1      #
806      * 1      # MAIN PROGRAMME
807      * 1      #
808      1      */
809      1
810      1      /* initialisation */
811      1      num_minterms = 0; num_dont_cares = 0; num_terms = 0;
812      1      new_data = true;
813      1
814      1      /* main loop */
815      1      DO WHILE (true);
816      2      /* get menu selection */
817      2      menu:
818      2      PUT SKIP(3);
819      2      CALL print_menu;
820      2      GO TO menu_option(menu_selection());
821      2
822      2      menu_option(1): /* enter data */
823      2      CALL enter_data;
824      2      GO TO menu;
825      2
826      2      menu_option(2): /* minimise */
827      2      IF num_terms < 2 THEN
828      2      PUT SKIP LIST ('Insufficient data - cannot minimise. ');
829      2      ELSE DO;

```

```
830      3      IF new_data THEN CALL run_minimisation;
831      3      PUT SKIP(2);
832      3      CALL output_results(SYSPRINT);
833      3      new_data = false;
834      3      CALL continue_prompt;
835      3      END;
836      2      GO TO menu;
837      2
838      2      menu_option(3): /* file results */
839      2      IF ^ new_data THEN DO;
840      3      OPEN FILE(results_file) TITLE ('BOOL_MIN -APPEND') LINESIZE(80) STREAM OUTPUT PRINT;
841      3      CALL output_results(results_file);
842      3      PUT PAGE FILE(results_file);
843      3      CLOSE FILE(results_file);
844      3      PUT SKIP LIST ('Results appended to file BOOL_MIN. ');
845      3      END;
846      2      ELSE PUT SKIP LIST ('No results to file. ');
847      2      GO TO menu;
848      2
849      2      menu_option(4): /* information */
850      2      CALL print_information;
851      2      CALL continue_prompt;
852      2      GO TO menu;
853      2
854      2      menu_option(5): /* quit */
855      2      STOP;
856      2      END;
857      1
858      1      END BSc_project;
```