Help for QMC2

down-top

down - up - top

A General Description

A.1 Functionality

The QMC2 program performs optimization of two-stage (AND/OR) logic by using the Quine-McCluskey and related algorithms.

It is assumed that the logic construction has n binary inputs which will lead to m outputs. Both input and output variables can be either “true”, “false” or “don’t care”. This program uses 1 for “true”, 0 for “false” and “-” for “don’t care”.
In terms of electronic circuits 1 and 0 would be described as “Hi” and “Lo” while “don’t care” means any value or not connected.

Any such logic construction can be described as two-stage logic, where for the first stage input values get combined by logic AND functions, leading to product terms (PT). In the second stage several of those PTs get combined by logic OR functions producing the outputs.
The inputs can be used in the normal or complementary (i.e. inverted) form. This type is called a disjunctive logic form.

The optimizing algorithms usually lead to a more compact description of the logic by removing redundancies that may be present in the input files. Then the logic construction has a lower complexity than before the optimization.

The program performs the following steps:

- **Loading or editing a logic description. Different formats of truth tables or boolean logic are available.**
- **Expanding the set of logic statements. For n inputs a maximum of 2^n Minterms (MT) are generated.**
- **Generation of Prime Implicants (PI) by either the Quine-McCluskey or the Consensus algorithm.**
- **Selection of a cover of MTs by the minimum PIs by heuristic methods.**
- **Bundle optimization. Multiple iterations of the ROLEXA algorithm can be chosen.**
- **Tabular output of the final result in bundle form.**
- **Mapping of the optimized output to logic AND, OR and INVERT gates.**
- **Automatic saving of the work area in ASCII format (.rpt) and in RichText** format **(.rpt.rtf)**

The resulting boolean formulae may be directly implemented in an electronic circuit, although no further mapping to other gates or to multi-stage logic is performed here.
The resulting bundle form can be directly implemented as a PLA circuit element. In case of a NOR-NOR PLA the inputs and outputs all have to be inverted.
The fully expanded form, as it is used for the final comparison, may be seen to represent a memory (ROM or RAM) with fully decoded n inputs and $\mathbf{2^n}$ words with a width of m bits each. In this case any gain in complexity is lost, though.

A.2 Program Version

QMC2 V1.4 - by GV

Compiled and tested with PwrDev and PBCC under Windows 10.

This software is freeware. There is no guarantee for correct results. Use at own risk.

down - up - top

B Usage

B.1 Starting the Program

Start by

- clicking the QMC2 icon on a Windows desktop
- or by dragging an input file over the QMC2 icon
- or by command line: **QMC2[.exe] [name]** <CR>

name may contain drive, path and/or file specifications. A file name has to be entered with a qualifier (extension), e.g.: test5.tab.
For any batch processing the **Display Level** is preset to 0.

down - up - top

B.2 Display

The size of the application area will be adapted to the available screen size automatically at start-up.
The character size in the edit and display window can be changed with the **TextSize Buttons [<<] and [>>]**.
The ASCII content of the display screen gets copied to the Clipboard by **[Text --> Clip]**.
At the left the progress of the single steps is reported in Progress Boxes.
An activity display at the bottom left indicates the current execution. At the end of each step the total CPU time is displayed.

B.3 Settings

Polarity	**1-Set :**	Use the **normal** (true) output functions (1 plus don't care values).
	0-Set :	Use the **inverse** of all output functions (0 plus don't care values).
	Automatic :	**Automatic choice** of each function's polarity for minimum P_Terms.
	Combination:	Calculation of **all possible** output polarities and choice of best result. Warning: It may take a long time.
Prime Generation	**QMC :**	**Quine-McCluskey** method of repeated comparisons for generation of all Prime Implicants QMC is only used for up to 10 input variables.
	CNS :	Recursive generation of all Prime Implicants by use of the **CONSENSUS** method.
ROLEXA	**0 :**	No bundle optimization used
	1 - 9999 :	Enter a number to increase the ROLEXA search time. This option is only used for multiple output functions. **Warning**: It may take a long time with numbers > 10.
Display Level	**0-Minimum :**	Show only the final result in bundle format and algebraic outputs.
	1-Results :	Minimum + results of each phase in the flow.
	2-In-Outputs :	Results + inputs and outputs of each step.
	3-Main-Steps :	In-Outputs + main intermediate steps.
	4-Details :	Main-Steps + more details. Warning: may be quite long

B.4 Execution

- **Load File** Existing files may be directly loaded on to the work area by the **Open Button** or **File | Open – Ctrl+O.**
  Either use them directly or edit and save them.
- **New Text** A new text may be generated and save under a new name by the **New Button** or **File | New File – Ctrl+N.**
- **Save Text** Any changed text has to be saved by the **Save Button** or **File | Save File – Ctrl+S** or by **Save AS...**
- **Start** The optimization begins by the **Start Button** or **Execution | (Re-)Start/Continue – Ctrl+G.**
- **Display** As long as the **Page-by-Page – Ctrl+B** mode is on a display stops it is longer than one page – depending on font size.
  The **Continue Button – Ctrl+G** will re-starts the display, which may continue until the current step is finished.
  While the **Continue Button** is enabled, some settings may still be changed.
  The **Interrupt Button – Ctrl+Q** will stop long displays and continue with the next step.
  Note that the **Interrupt** and **Continue Buttons** cannot be moved with the main window.
- **Break** The operation will be stopped by the **Break Button** or **Execution | Break / Interrupt / Quit – Ctrl+Q.**
  No report files will be generated in this case.
- **Re-Start** When all steps have been completed – or after a **Break** the last input file will be reloaded for repeated or modified execution:
  (Re-)Start Button or Execution | (Re-)Start/Continue – Ctrl+G.
- **Quit** The execution stops and the program gets closed by the **Quit Button or Execution | Break / Interrupt / Quit – Ctrl+Q.**
- **Emergency** An extra **Emergency Console** task exists that allows the user to close any frozen or hanging program by
  Clicking [X] or Ctrl+Q. The task's display mode can be changed between **Background (B)** to **Always on Top (T)**.

B.5 Report Files

All data displayed on the work area are recorded in a

***.rpt**	output file containing just **ASCII** text information and in a
***.rpt.rtf**	file in **RichText** format, containing also the font, text size and colour information. It can be reviewed in standard word processors.

When a run has finished, the report files can be copied to other names or locations by the **File | Save As...** menu command.

down - up - top

C Formats

C.1 Size Limitations

Maximum number of **inputs** (CNS) :	22
Maximum number of **inputs** (QMC) :	10
Maximum number of **outputs** :	100

C.2 Input Tables and Boolean Equations

All input formats (table and boolean equations) are converted into the internal **truth table format**.
Names of input and output variables are automatically generated when the input file does not provide them (possible for *.pla).

Consistency checking of inputs is done for table, but not for boolean entries. Boolean entries have to be in disjunctive normal form.
There is no algebraic manipulation. Multiple boolean assignments to one output function result in OR-ing of the entries.

The inputs are completely specified when all 2^n possible combinations of input terms appear. Incompletely specified means that some input combinations are missing. Some formats allow a **$Rest** statement to assign output values to the non-specified input terms (*.dcb, free format).
QMC uses 0 (false) as a default for the functions of all undefined input combinations.

There may be **more than 2^n entry lines** as long as they describe consistent data sets.

The input fields (**X-Inputs**) may contain **1 (true)**, **0 (false)** and **- (don't care = DC)**. A DC input value is an abbreviation for a 1 and a 0. Thus, m DC inputs represent 2^m input terms.
An error will be indicated when the expanded input terms produce non-consistent (contradictory) output values.

The function fields (**Y-Outputs**) of a truth table may also contain **1**, **0**, and **- (DC)**.
An output DC value represents a 1 or a 0 but the user does not care which one. After optimization it will be defined to be either a 1 or a 0.
Boolean equations cannot represent output don't cares, though.

Example of a **truth table** (3 input variables a,b,c and 4 output functions w,x,y,z) and the corresponding **expanded** form:

```
Input:  a b c | w x y z                          Expanded:  a b c | w x y z
        ------+--------                                     ------+--------
        0 0 0 | 1 0 - 1   <-- e.g. output DC                0 0 0 | 1 0 - 1
        0 0 1 | 1 - 1 0                                     0 0 1 | 1 - 1 0
        0 1 0 | - 1 0 1                                     0 1 0 | - 1 0 1
        1 1 - | 0 0 1 1   <-- e.g. input DC                 0 1 1 | 1 1 0 -   <- from $Rest
        $Rest | 1 1 0 -   <-- optional                      1 0 0 | 1 1 0 -   <- from $Rest
                                                            1 0 1 | 1 1 0 -   <- from $Rest
                                                            1 1 0 | 0 0 1 1
                                                            1 1 1 | 0 0 0 1
```

C.3 Truth Tables Formats

(F1) *.dcb	subset of the LOGIC2 *FUNCTIONAL-TABLE format (TM)
(F2) *.tdf	subset of the Altera AHDL TABLE format (TM)
(F3) *.pla	subset of format as used by ESPRESSO 2.3 from UC Berkeley
(F4) other qualifiers	free table format, e.g. *.tab

Examples of truth table inputs:

(F1) *.dcb (LOGIC2 table):

```
*IDE
    ... any text ...          (optional)
 *INT                         (optional)
    IN : a,b,c;               (optional)
    OUT : x,y,z;              (optional)
 *FUN
    $HEADER: X [a,b,c] : Y [x,y,z]
    X   000   : Y  10-;       (- for don't care in inputs and outputs)
    X   001   : Y  0-1;       (; before comments)
    X   011   : Y  -11;
    X   1--   : Y  011;
    X  $REST  : Y  -10;       ($REST is optional)
 *END                         (optional)
```

down - up - top

(F2) *.tdf (AHDL table):

```
SUBDESIGN <filename>
( a,b,c   :INPUT;            (optional)
  x,y,z   :OUTPUT;)          (optional)
BEGIN
TABLE
  (a,b,c) => (x,y,z);        (do not use / in names)
   B"000" => B"100";         (; or -- before comments)
   B"001" => B"101";         (no don't cares in outputs!)
   B"010" => B"010";
   B"011" => B"001";
   B"1xx" => B"100";         (x for don't care in inputs)
END TABLE;                   (no 'REST' assignment possible!)
END;                         (optional)
```

(F3) *.pla (ESPRESSO table):

```
# pla input file             (comment)
.i 3                         (number of inputs)
.o 4                         (number of outputs)
.ilb a0 a1 a2                (input names, optional)
.ob y0 y1 y2 y3              (output names, optional)
.p 3                         (number of product terms)
.type fd                     (table type: f, fd (default), fr, fdr, optional)
  000 0110
  0-1 1101
  10- 0--1
  1-1 1~0~                   (~ never has any meaning)
.e                           (end of file)
```

According to the **ESPRESSO 2.3** definitions the identifier .type controls the interpretation of the table entries.
The meaning of output characters is:

.type	1	0	-	undefined terms	
f	ON	none		OFF	no -, no DC
fd	ON	none	DC	OFF	overlap of ON and DC => DC (default of .type)
fr	ON	OFF	none	DC	real truth table with default = DC
fdr	ON	OFF	DC	none	no others

(F4) any other qualifiers (free table format, e.g. *.tab):

```
a,b,c : x,y,z                (: or => for In-Out-separator)
      000    : 10-  ;... text ...   (; before comments)
      001    : 0x1                  (x or - for output don't care)
      011    : -11
      0x1    : 01-                  (x or - for input don't care)
      1--    : 011
      $REST  : -01                  (has to be the last line, optional)
```

C.4 Boolean Equation Formats

(F5) *.dcb	subset of the LOGIC2 *BOOLEAN-EQUATION format (TM)
(F6) *.tdf	subset of the Altera AHDL boolean format (TM)
(F7) other qualifiers	free boolean format, e.g. *.bol

Note: Boolean entries have to be in disjunctive normal form. There is no algebraic manipulation.

(F5) *.dcb (LOGIC2 boolean):

```
*IDE
... any text ...                   (optional)
*INT                               (optional)
IN:a,b,c;                          (optional)
OUT:x,y,z;                         (optional)
*BOO
y =  a & /b &  c + /a & b & /c
z = /a &  b & /c                   (AND: &;  OR: +;  NOT: /)
  +  a & /b &  c;                  (continuation possible with OR term)
*END                               (optional)
```

down - up - top

(F6) **_*.tdf (AHDL boolean):_**

```
SUBDESIGN <filename>
 ( a,b,c   :INPUT;                 (optional)
  x,y,z  :OUTPUT;)                (optional)
BEGIN
 y =  a & !b &  c # !a & b & !c;   (do not use / as part of input or output names!)
 z = not a and  b or !c            (AND: &, and;  OR: #, or;  NOT: !, not)
   #  a & !b &  c;                 (continuation possible with OR term)
END;                               (optional)
```

(F7) **_other qualifiers (free boolean format, e.g. *.bol):_**

```
y =  a * /b + a and not c          (AND: *, &, and;  OR: +, #, or;  NOT: /, !, not)
z = !a &  b # a & !c               (*, & = AND, +, # = OR, /, ! = NOT)
  +  a * /b *  c                   (continuation possible with OR term)
  or not a and b
```

C.5 Bundle Output Format

The results are shown in a compact **bundle format** (not to be confused with the truth table format).
The input field uses **1**, **0** and **-** with the **same meaning as for truth tables**.
The DC (-) is not counted as an X_Literal.

In the output field a **1 means a normal output,** a
0 means an inverted output and a
dot (.) that the input term is not used at all.

Note that an output function (a column) has either 1's or 0's (and dots). This sequence of 1's and 0's across is called the **Polarity** of the output functions. The dot (.) is also not counted as a Y_Literal.

An output function is '**trivial**' (1 or 0), when there are only don't cares in the X-entry line. In such a case, the line is not counted as a P_Term and no Y_Literals are counted.

An output function is '**not defined**' when all output terms of the function are don't care dots (.).

In SOP (Sum-Of-Products) terms, the **X_Literals** represent the inputs for **AND gates** per input term and the **Y_Literals** indicate which AND outputs are inputs to an **OR gate** per output function.
The bundle format is thus directly equivalent to **PLA placement** or to an **AND/OR logic circuit**.

When a Display level '**2-In/Outputs'** or higher gets chosen, the final expanded result will get **compared to the (expanded) input table**.
The expanded result then shows how **output don't cares of the input file are finally chosen**.
The results are also shown as boolean equations with the necessary number of multiple input **AND and OR gates** and the total number of necessary inputs.
At the end of the bundle output a summary shows the total of X_Literals, Y_Literals and P_Terms. The value **Gain** is the number of literals for true functions in the expanded form - see Expand Input (4) - in relation to the number of final literals after optimization.

Example of a bundle output (optimization of above example) and the finally expanded form with **Polarity = 1 1 0 1**
(output y has to get inverted):

```
Optimized:   a b c | w x y z                 Expanded:   a b c | w x y z
             ------+--------                             ------+---------
             0 - - | 1 . . .                             0 0 0 | 1 0 1 1
             - - 0 | . . . 1                             0 0 1 | 1 0 1 0
             1 0 - | 1 1 0 .                             0 1 0 | 1 1 0 1
             0 1 - | . 1 0 .                             0 1 1 | 1 1 0 0
             1 - - | . . . 1                             1 0 0 | 1 1 0 1
                                                         1 0 1 | 1 1 0 1
         X_Lits=7  Y_lits=8  P_Terms=5                   1 1 0 | 0 0 1 1
                                                         1 1 1 | 0 0 1 1
```

Note.: When expanding the bundle output after optimization, each input DC gets replaced by a 1 and a 0. By this expansion, an output dot may get overwritten (covered) by another input term (either by 1 or by 0 depending on the polarity). At the end, all remaining dots are overwritten by the inverse polarity (0 for polarity 1 and 1 for polarity 0).

C.6 Examples

A number of Examples with different input formats and different number of elements are provided for testing purposes.

Note: Some of the input files lead to very long calculation times – especially when choosing the **0-Set**, **Automatic** or **Combination** settings.
At Display Levels > 1 it can be observed at which step the delays take place.

down - up - top

D Algorithms

D.1 Input Processing

The table formats are basically **reformatted** to correspond to the **internal truth table format**. Boolean inputs are first scanned on the left side for the number and names of output variables and then on the right for the number and names of input variables. If a variable neither appears in an equation in non-inverted nor in inverted form, a don't care (DC) is generated in the truth table.

D.2 Expansion of Inputs

In a recursive process all input DC are expanded to a **1** and subsequently to a **0**. When an expanded input term A already exists and B gets generated with the same input sequence by expansion of another input line, the outputs are merged according to the following table. If two outputs have opposite values, an Error is reported.

A B	1	0	-
1	1	Err	1
0	Err	0	0
-	1	0	-

`Err   means Error (contradiction in the definition)`

D.3 Generation of Prime Implicants

The **Prime Implicants** (PI) are generated for each output function separately. Both, QMC and CNS will generate **all possible** PI per function.

For the generation of implicants the inputs terms are used that lead to a **1** or DC for functions with polarity 1. For functions with polarity **0** terms are selected that produce a 0 or DC. The consideration of the **DC-Set** (input terms that lead to a don't care) may lead to more Implicants containing possibly more DC terms.

Note: The number of Prime Implicants can be quite large. For n inputs there may be up to **3^n/n PI**.

(a) QMC - Quine-McCluskey

The QMC method compares each input term with every other one. When two terms differ in only one position where one has a 1 and the other a 0 in that position, **the two source terms get replaced by a single term don't care** in that position. The same gets applied when one of the differing entries is a don't care.

```
Example:  0-100
          0-110    => Reduction by replacing 0 and 1 by -
          -----
          0-1-0    => Resulting single term has one more don't care
```

The procedure is **iterated until no changes occur** any more.

The set of input terms gets sorted into groups where the number of 1's differs by one, i.e. at first into a group without any 1's then into a group with a single 1 then a group with two 1's etc. CPU time is saved by only comparing the members of one group to the next one. After each iteration step the duplicate terms are removed and any term gets added to the resulting set that has not participated in a reduction.

(b) CNS - Consensus

CNS works recursively. On each level a splitting variable is chosen where most 1's and 0's appear for an input variable. The set is divided into a subset **1-Cofactor** with the splitting variable equal to 1 and a subset **0-Cofactor**. Each subset is treated separately by setting the splitting variable to don't care. This procedure continues until all terms are unate (i.e. all non-split variables contain only 1's or only 0's).

The replaced 1's and 0's are re-inserted again and the procedure returns both Cofactors one level up in hierarchy where CONSENSUS and SCC take place.

First **CONSENSUS** compares each term of the 1-Cofactor to each line of the 0-Cofactor.

A B	1	0	-
1	1	/	*
0	/	0	*
-	*	*	-

`*       means a possible DC position`

`/       means no cover possible`

When there is exactly one * at the end of the comparison, then a new line is created in the (third) C-set with the * replaced by – (DC).

Secondly, a **SCC** (Single Containment Cover) comparison process starts to compare each term of the C-set to all terms in two Cofactor sets.

A and B are the elements of two terms:

```
   A | 1 | 0 | - |
 B   |   |   |   |
-----+---+---+---+
 1   | = | / |A>B|          =      means identical
-----+---+---+---+
 0   | / | = |A>B|          /      means no cover possible
-----+---+---+---+
 -   |B>A|B>A| = |          A>B    means A covers B
-----+---+---+---+
```

When a comparison returns all positions identical (=) one term can be dropped (duplicate). When a / or both A>B and B>A occur the comparison can be stopped without success. One term covers another one if only A>B (or B>A) occurs (at least once) in the comparison. The covered term can then be dropped from its set.

After re-inserting 1's (or 0's) in the splitting position, the recursion returns one level up where CONSENSUS and SCC are performed again. The procedure ends when the top is reached.

Two special cases can accelerate the computations:

(1) There are two lines left after splitting. They can return immediately to the SCC/CONSENSUS level.

(2) A Cofactor set contains a full binary set in some variables. This results in one term with all variables set to – (DC).
This one term also gets returned directly.

D.4 Covering Process

For the covering phase a **Cover Matrix** is built up with the **Minterms** of the target function (no DC-set any more) in rows and all **Prime Implicants** that cover those Minterms in columns. The matrix elements get a tag at the crossing point of a MT and a PI if the PI covers the MT. The selection of the minimum set of PI to cover the Minterms uses a three step approach.

(1) When a MT is **covered by only one PI,** then that PI is called an **Essential Prime (ESS)** and may of course cover other MT.
The covered Minterm is removed from the matrix as well as the Essential Prime which is stored as part of the solution. Of course all other MT covered by this ESS are also removed from the matrix. This step is repeatedly applied until no Essential Primes can be found any more.

(2) **Secondary Essential Primes (SEC)** are those that are left with one PI cover after phase (1). They also get removed and stored as part of the solution.

(3) In the case that all MT are covered by more than one PI, the problem has a cyclic character (several solutions possible).
The next **Alternative Secondary Prime (ALT)** is chosen after calculating a weight factor for each Prime. First a weight is calculated for each of the Minterms as the inverse of the number of PI covering this Minterm (thus MT are preferred which have few covers). Now the Minterm weights are summed up per Prime Implicant and the largest one gets chosen (thus PIs are preferred that cover many MT). This ALT Prime is then removed and stored as solution.

After each removal of a PI and the corresponding Minterm(s) an iterated check of **row and column dominance** takes place. If a PI covers the same MT as (or more than) another PI, the latter can be removed (the column with more covers is kept). On the other hand a row with fewer MT covers is kept, while the one that gets covered by the same and more PI is removed.

D.5 Automatic Polarity

For this option the selection of Prime Implicants is done for **all functions** firstly with normal polarity (1) and secondly with the inverse polarity (0). In a first simple step the polarity is chosen with minimal P_Terms for each function. The common input terms are weighted differently in further calculations, so that the total P_Terms are minimized.

D.6 Bundle Processing

For further reduction of the number of P_Terms as well as X_ and Y_Literals the **ROLEXA algorithm** is applied (Recursive One Level Expansion and Absorption). Iteratively 3 main steps are performed:

(1) Expansion of single input don't cares by replacing them with 1 and subsequently with 0. A search is started to find other input terms that cover the expanded terms. When both are found the original term may be dropped while the output terms get merged.

down - up - top

(2)   Absorption of redundant output terms: When one term covers a second one after a SCC check, common Y_Literals can be removed from the output term of the covered input term:

```
 input   |  output              =>         input   |  output
--------+----------                     --------+----------
- 0 1 - | 1 . . 0 .                     - 0 1 - | 1 . . 0 .
1 0 1 0 | 1 . 1 . 1                     1 0 1 0 | * . 1 . 1     <=   * (removed 1) becomes .
```

(3)   Reduction of outputs for single bit input redundancy: If now two input terms differ in one variable (one is 1, the other 0) and one output term is a true subset of the other one, then the input position of the subset gets a - (DC) and the output positions get removed (.) in the superset:

```
 input   |  output              =>         input   |  output
--------+----------                     --------+----------
0 1 1 - | 1 . 0 . 1        (subset)     - 1 1 - | 1 . 0 . 1     <=   * (removed 0) become   -
1 1 1 - | 1 . 0 . .       (superset)    1 1 1 - | 1 . 0 . .
```

The method is order dependent and can be repeated with statistically varying starting points in (1) with the Bundle repetition factor (0..9999).
A final step Bundle Cover erases possible redundant entries in the output terms. During '**Automatic**' and '**Combination.**" mode, the 1-set and the 0-set are also optimized by Bundle Cover before mixed polarities are calculated.

D.7 Comparison with Expanded Inputs

During this step the resulting bundle list is fully expanded w.r.t. the inputs and the output filled with the generated 1 and 0 values.
All remaining dots (.) get filled with the inverse value of the output polarity. The line by line comparison with the original fully expanded input table reveals how the output don't cares get generated by this solution. The following function types get displayed:
Trivial, **Undefined**, **Identical** and **Inverse**.

Note.: An inverse function may possibly only be detected after comparison of the expanded results.

D.8 Boolean Equations and Gates

Product terms appearing more than once are generated as **PT_n**, while AND-ing the relevant inputs. In summary, the number and width of the corresponding AND and OR gates are counted. Without further mapping to any library elements, the total number of gate inputs is also displayed.

D.9 Literature

G. De Micheli:	Synthesis and Optimization of Digital Circuits McGraw-Hill 1994
G. D. Hachtel, F. Somenzi:	Logic Synthesis and Verification Algorithms, Kluwer Academic Publishers, 1996
P. Molitor, C. Scholl:	Datenstrukturen und effiziente Algorithmen fuer die Logiksynthese kombinatorischer Schaltungen, B.G. Teubner, 1999

up - top