NAME

dw set macro macro-name macro-expr – creates a new user-defined macro for use with subsequent **dw set assert** commands

DESCRIPTION

The dw constraint compiler allows the user to enter assertions via the **dw set assert** command, which are expressions built using the Bash shell expression syntax. Additionally, **dw** includes built-in and user-defined macros which may be invoked as assertions. **The dw set macro** subcommand creates new user-defined macros.

SYNTAX

The **dw set macro** subcommand has two required arguments. The first is the macro name and the second is the macro expression. The macro name is any sequence of letters, digits and the character '_' (underscore), where the first character is not a digit. The macro expression is a Bash shell expression using $1, $2, … for its arguments. Since the Bash shell will interpret $n as a positional character, the definition of the macro expression should be enclosed in single quotes to prevent interpretation. Note that it is not necessary to list the number of arguments required by the macro – the argument list is inferred from parsing the $n arguments in the macro expression.

Each invocation of **dw set macro** appends to the file user-macro.m4 in the current dw workspace. The first invocation of dw set macro following dw init creates this file. The entire set of macros available at any point during dw constraint compilation consists of the built-in macros in $DWAVE_HOME/bin/dw-macro.m4 and any user-defined macros in user-macro.m4 in the current workspace. Display this list via the dw get macro command.

Both built-in and user-defined macros are maintained internally using the m4 macro language. Each invocation of dw set macro is converted into an m4 command appended to the user-macro.m4 file in the current workspace.

The naming convention for built-in macros consists of three parts: a type specifier, argument number information and a textual description. The type specifier is either the letter 'R' (Relation) or the letter 'F' (Function). The distinction is that

in ‘F’ type macros some arguments have functional dependence on other macro arguments. An example is the **F2_1_and** macro, since the valid states of an assertion built using this macro are those in which the third argument is the logical AND of the first two argument. Macros which have no functional dependence use the ‘R’ type specifier. An example is the **R2_nand** macro, whose valid states are those in which the logical NAND (Not AND) of its two input arguments is true.

‘F’-type macros are followed by a pair of numeric arguments. The first specifies the number of input arguments and the second specifies the number of output arguments. The two arguments are separated by the character ‘_’ (underscore). ‘R’-type macros are followed by a single numeric argument, which is the total number of arguments to the macro.

The last portion is a textual description of the macro – either relational or functional. This field is separated from the argument number information by the character ‘_’ (underscore).

LIST

R1_on – valid state consists of its single argument set to 1

R1_off – valid state consists of its single argument set to 0

R2_eq – valid states are when its two arguments are equal

R2_ne – valid states are when its two arguments are not equal

R2_lt – valid states are when its first argument is less than its second argument

R2_le – valid states are when its first argument is less than or equal to its second argument

R2_gt – valid states are when its first argument is greater than its second argument

R2_ge – valid states are when its first argument is greater than or equal to its second argument

R2_or – valid states are when the logical OR of its two arguments is true (1)

R2_nand – valid states are when the logical NAND (Not AND) of its two arguments is true (1)

F2_1_and – valid states are when the third argument is the logical AND of the first two arguments

F2_1_or – valid states are when the third argument is the logical OR of the first two arguments

F2_2_half_adder – valid states are when the third and fourth arguments are the sum and carry bits of the sum of the first and second argument

F3_2_full_adder – valid states are when the third and fourth arguments are the sum and carry bits of the sum of the first three arguments

R3_one_of_n – valid states are when exactly one of the three arguments is set to 1

R4_one_of_n – valid states are when exactly one of the four arguments is set to 1

R5_one_of_n – valid states are when exactly one of the five arguments is set to 1

R3_two_of_n – valid states are when exactly two of the three arguments are set to 1

R4_two_of_n – valid states are when exactly two of the four arguments are set to 1

R5_two_of_n – valid states are when exactly two of the five arguments are set to 1

EXAMPLE

To turn logical variable **x** on, use the **R1_on** macro in a **dw set assert** command:

```
dw set assert 'R1_on(x)'
```

The assertion must be enclosed in single quotes so that the shell does not interpret the parentheses as a control operator or metacharacter. The invocation of R1_on(x) expands to x-1, which is a linear expression that is squared by the dw constraint compiler to form the QUBO term 1-x. This evaluates to 0 and is hence valid when x is 1 and otherwise evaluates to a positive value.

Define a new user-defined macro like this:

```
dw set macro mymacro '$1 + $2 - $3'
```

After this command, use the newly defined macro in a **dw set assert** command as follows:

```
dw set assert 'mymacro(x,y,z)'
```

This macro defines valid states for the three variables to be those in which the sum of the **x** and **y** variables equals the **z** variable.

BUGS

Please report bugs to dwsupport@dwavesys.com.

COPYRIGHT

SEE ALSO

dw(1), m4(1)