Floating-point library for PIC18 processors

version 1.03

This system library is available in two versions – both were prepared as a direct replacement of the official floating-point library (__Lib_MathDouble.mcl) with removed problems that were inherited from the original Microchip library. Second version additionally introduces math error exceptions. (I could not use the **try .. except** keywords as these are reserved by the compiler, but, effectively, similar program behaviour is possible.) The library produces significantly smaller code than the official one and gives more accurate calculation results.

Problems that were corrected:

- lack of unbiased rounding,
- lack of rounding while converting to integer types,
- lack of big numbers 'rounding', i.e. conversion error reduction,
- max negative number reported as integer overflow,
- overflow for division by zero does not reflect sign of dividend,
- no protection against numbers with EXP=0 and significand<>0 that do not belong to Microchip format.

The so-called unbiased rounding leads to increased calculations accuracy by forcing residual errors to be 'unbiased', i.e. non-cumulative. It's lack of rounding that is mostly responsible for the trailing 9s while converting real numbers to strings in the official library.

Added library features:

- exception mechanism for f-p math errors,
- f-p math operations status available with Get_FPstatus function,
- minreal, maxreal & epsreal constants.

There are just four public routines in MathDouble library (click on them to see their prototypes):

Clear_FPstatus
Get_FPstatus
FPerror
FPraise

Status of floating-point operations may be cleared with Clear_FPstatus procedure and read with function Get_FPstatus. Available bits of the status byte are declared as constants:

FP_IOV - integer types overflow flag
FP_FOV - floating point overflow exception flag
FP_FUN - floating point underflow exception flag
FP_FDZ - floating point divide by zero exception flag
FP_NAN - not-a-number flag

FP_DOM - domain error flag

as well as values of the status byte in specified situations:

FPS_IOV - integer types type overflow
FPS_FOV - floating point overflow
FPS_FUN - floating point underflow
FPS_FDZ - floating point divide by zero (and overflow)
FPS_NAN - not-a-number
FPS_DOM - domain error

Boolean function FPerror is used for error testing in the exception mechanism. Exception is raised automatically in case of overflow, underflow, or division by zero. One may raise an exception using FPraise procedure with mask=0. Calling FPraise with mask containing any combination of FPS_IOV, FPS_NAN, or FPS_DOM, will ensure exception if any of corresponding events that arise before the call.

Integer type overflow or domain error may be detected during conversion from real to integer types. Calculation routines automatically correct real numbers with zero exponent and non-zero mantissa (that do not belong to Microchip format) without setting the domain error flag. Not a Number flag may be set during number conversion from IEEE 754 format to Microchip format or in case of undefined result of some math functions, like sqrt or pow.

Exception mechanism may be used in a following way:

Code:

```
var status: byte;
    st: string[20];
    x,y: real;

  if not FPerror then
   begin
    y:=x/y;
    if y<0.0 then FPraise(0);     // raise user-defined exception
    st:='success';
   end
  else
   begin
    stat:=Get_FPstatus;
    case stat of                  // determine exception cause
     0:       st:='negative result';   // user-defined exception
     FPS_FOV: st:='overflow';
```

```
        FPS_FUN: st:='underflow';
        FPS_FDZ: st:='division by zero';
       end;
      end;
     Clear_FPstatus;                  // deactivate exceptions
```

If one converts numbers of type real to any integral type (whether explicitly or implicitly), adding Fpraise(FPS_IOV) after conversion(s) will ensure raising an exception in case of this integral type overflow. Similarly, Fpraise(FPS_NAN) may take care of signaling problems with conversion from IEEE 754 to Microchip format or with sqrt and pow calculations.

Another example:

Code:

```
    procedure handle_FPerrors;
     begin
      // any necessary steps in case of fp-math error
      ...
      calc_error:=true;
      Clear_FPstatus;      // deactivate fp exceptions & clear status
     End;{handle_FPerror}

    begin
     ...
     calc_error:=false;
     if FPerror then handle_FPerrors
      else
       begin
        int_var:=integer(x/y);
        FPraise(FPS_IOV);     // raise exception if integer overflow
       end
```

The exception mechanism ensures that the conditional **if** FPerror **then** ... will be executed when an error occurs, even though it precedes the calculations.

And finally, somewhat less elegant solution:

Code:

```
     calc_error:=false;
     if FPerror then goto label1;
     y:=x/y;
     ...
     goto label2;
    label1:
     // any necessary steps in case of fp-math error
     ...
     Clear_FPstatus;                  // deactivate exceptions
     calc_error:=true;
     ...
    label2:
```

As mentioned previously, the MathDouble lib comes in two versions. When using the version with

exception mechanism, remember to call the Clear_FPstatus procedure at program beginning. Naturally, one does not have to always use exception mechanism with the version equipped with it – without activating the mechanism, calculations will be performed just like with the other version.

Due to introduction of rounding while converting to integer types (in both implicit and explicit conversion), use of the replacement lib may require older code correction, as with the official lib conversion was based on truncation (the fractional part of real number was simply omitted). This issue is addressed by introducing a set of useful routines to the Trigon library – one of them, Trunc, may be used whenever rounding is not wanted. There is also Round function which does nothing more than explicit conversion for variables but should be used with constants or there will be a difference between runtime and compile-time calculations.

When using the MathDouble replacement lib, one should also use the other replacement libs – both the floating-point ones, Trigon and Trigonometry, and the Strings and Conversions libraries, on which some floating-point routines depend.

Both str2float and Float2str (or FloatToStr) procedures from the Conversions library replacement may still be used, though the new str2real and Real2str procedures from Trigon lib replacement are more advanced and Real2str is much faster.

IMPORTANT: Starting from v. 1.00, the library requires presence of Math library replacement.

NOTE: When using the MathDouble lib version with the exceptions mechanism, software simulator may stop on assembly code changing STKPTR that is part of the exception mechanism – in such case one may use Step_Out to go back to Pascal code. Another solution is to use breakpoints to jump over code that may cause f-p math error. Naturally, all works fine in real environment.

Manual library installation:

- find mP PRO installation directory and subdirectory **.../Uses/P18**,
- find original library file **__Lib_MathDouble.mcl** there and rename it,
- unpack the replacement lib – there'll be two versions:

  __Lib_ MathDouble_exc.mcl

  __Lib_ MathDouble_no_exc.mcl

  choose one of them, change it's name to **__Lib_ MathDouble.mcl** and move the file to the **.../Uses/P18** directory.

Have fun,
janni

MathDouble library - system fp-math routines for PIC18 processors

version 1.03
date 26.06.14

Revision history:

0.05 - modified for mP PRO 2.15, added status byte access
0.06 - added DOM test for conversion routines, simplified code
0.07 - added exception mechanism
0.08 – adjusted to Trigon and Trigonometry replacement libs
0.09 - FPstatus cleared in FPerror now, added FPraise
1.00 – combined with Math library
1.01 – compiled with mP PRO 5.0
1.02 – compiled with mP PRO 6.01, added minreal, maxreal and epsreal
1.03 – changed declarations of some constants

This is system library – most routines are not directly accessible. Those that are:

procedure Clear_FPstatus;
clears status of f-p math operations

function Get_FPstatus: byte;
reads status of previous f-p math operations

function FPerror: boolean;
used for fp-math error test; present only in the version with fp-math errors exceptions

procedure FPraise(mask:byte);
raises exception if error declared in mask took place before, or if mask=0 (user exception);
present only in the version with fp-math errors exceptions

A version string is declared that may be used in code for verification:

const Lib_MathDouble_ver: string[4]