AN2113 基于 EC30-EKSTM32 扩展浮点运算

CREATE: 2010/08/05
UPDATE: 2010/08/05
GUTTA Ladder Editor Version 1.1
Version 1.1
http://www.plcol.com
http://www.visiblecontrol.com

概述

基于 Cortex-M3 内核的 STM32F103 系列单片机，并没有浮点运算协处理器。在 STM32F103 上进行的浮点运算都是软件模拟实现。考虑到加入浮点运算库需要大约 10K 左右的 FLASH 空间（即<math.h>对应的数学库），而且浮点运算速度较慢，EC30-EKSTM32 的指令集不包含浮点运算以及相关的转换指令。

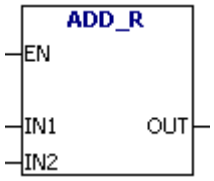
GUTTA 软件平台有对浮点数的完整支持。如果您希望 EC30-EKSTM32 支持浮点运算，可以借助 C 语言二次开发接口，在原 EC30-EKSTM32 系统上轻松的扩展浮点运算。借助本工程，您可以不用编写任何 C 代码，就让 EC30-EKSTM32 支持浮点运算，你只需要：

- 将项目提供的 **ManagerFun.xml** 文件替换掉 GUTTA Ladder Editor 软件安装目录下\GuttaLad\EC30-EKSTM32**ManagerFun.xml** 文件。
- 将项目提供的 **EC30-EKSTM32.hex** 程序文件通过 GUTTA Flash Utility 软件下载到 EC30-EKSTM32 内核中去。

只需要完成这两步，更新后的 EC30-EKSTM32 内核便支持浮点运算了。这里对这个开发项目做详细说明是为了大家深入理解 C 语言二次开发的工作方式，方便大家做其他扩展。

指令描述

+R

LAD	STL
	+R IN1, OUT

实数加法指令将两个 32 位实数相加，并产生一个 32 位的实数结果。

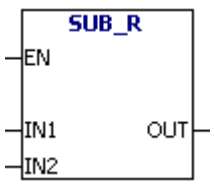
在 LAD 中：

$$OUT = IN1 + IN2$$

在 STL 中：

$$OUT = OUT + IN1$$

-R

LAD	STL
	-R IN1, OUT

实数减法指令将两个 32 位实数相减，并产生一个 32 位的实数结果。

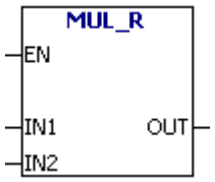
在 LAD 中：

$$OUT = IN1 - IN2$$

在 STL 中：

$$OUT = OUT - IN1$$

*R

LAD	STL
	*R IN1, OUT

实数乘法指令将两个 32 位实数相乘，并产生一个 32 位的实数结果。

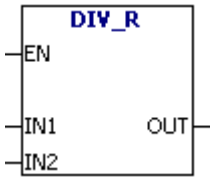
在 LAD 中：

$$OUT = IN1 \times IN2$$

在 STL 中：

$$OUT = OUT \times IN1$$

/R

LAD	STL
	/R IN1, OUT

实数除法指令将两个 32 位实数相除，并产生一个 32 位的实数结果。

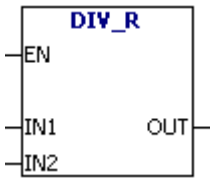
在 LAD 中：

$$OUT = IN1 \div IN2$$

在 STL 中：

$$OUT = OUT \div IN1$$

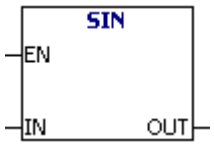
SQRT

LAD	STL
	SQRT IN, OUT

平方根指令对 32 位实数 IN 取平方根，并产生一个 32 位的实数结果 OUT。

$OUT = \sqrt[2]{IN}$

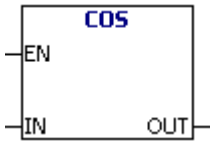
SIN

LAD	STL
	SIN IN, OUT

正弦指令对角度值 IN 进行三角运算，并将结果放置在 OUT 中。输入角以弧度为单位。

$OUT = \sin(IN)$

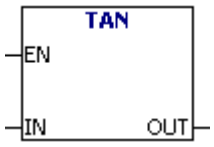
COS

LAD	STL
	COS IN, OUT

余弦指令对角度值 IN 进行三角运算，并将结果放置在 OUT 中。输入角以弧度为单位。

$OUT = \cos(IN)$

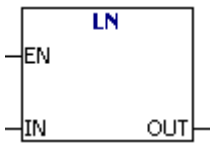
TAN

LAD	STL
	TAN IN, OUT

正切指令对角度值 IN 进行三角运算，并将结果放置在 OUT 中。输入角以弧度为单位。

$OUT = \tan(IN)$

LN

LAD	STL
	LN IN, OUT

自然对数指令对 IN 中的数值进行自然对数计算，并将结果放置在 OUT 中。

$OUT = \ln(IN)$

EXP

LAD	STL
	EXP IN, OUT

自然指数指令对 IN 中的数值进行自然指数计算，并将结果放置在 OUT 中。

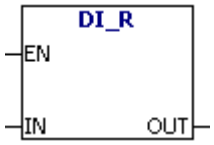
$OUT = \exp(IN)$

MOVR

LAD	STL
	MOVR IN, OUT

移动实数指令将 32 位实数从输入双字 IN 移至输出双字 OUT，不改变原来的数值。

DTR

LAD	STL
	DTR IN, OUT

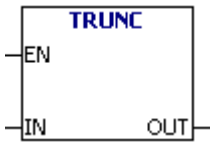
双整数至实数指令将 32 位带符号整数 IN 转换成 32 位实数，并将结果置入 OUT 指定的变量中。

ROUND

LAD	STL
	ROUND IN, OUT

实数至双整数（四舍五入）指令将 32 位实数 IN 转换成 32 位带符号整数，并将结果置入 OUT 指定的变量中。如果小数部分等于或大于 0.5，则进位为整数。

TRUNC

LAD	STL
	TRUNC IN, OUT

实数至双整数（舍去小数）指令将 32 位实数 IN 转换成 32 位带符号整数，并将结果置入 OUT 指定的变量中。只有实数的整数部分被转换，小数部分被丢弃。

实现代码

首先，我们需要在 GUTTA Ladder Editor 的指令系统中加入这些指令。加入这些指令可以通过修改 GUTTA Ladder Editor 软件安装目录下的

\GuttaLad\EC30-EKSTM32**ManagerFun.xml**

文件来实现。这个文件的具体描述可以参考《UM4003 指令描述文件规范》。这个文件即使是添加的内容，也超过 10 页，这里不打算列出，需要了解详情请参考项目中的 **ManagerFun.xml** 文件。

下面我们来看看二次开发项目中的 main.c 文件。

```
#include <math.h>
#include "system.h"
#include "stm32f10x.h"

void excute_ADDR    (void);
void excute_SUBR    (void);
void excute_MULR    (void);
void excute_DIVR    (void);
void excute_SQRT    (void);
void excute_SIN     (void);
void excute_COS     (void);
void excute_TAN     (void);
void excute_LN      (void);
void excute_EXP     (void);
void excute_MOVR    (void);
void excute_DTR     (void);
void excute_ROUND   (void);
void excute_TRUNC   (void);

uint32_t main(uint32_t action, uint32_t param) {

   switch (action) {
```

```
        case CUSTOM_LGC_INS_DISPATCH:
            switch (param) {
                case 280: excute_ADDR   (); return RT_OK;
                case 281: excute_SUBR   (); return RT_OK;
                case 282: excute_MULR   (); return RT_OK;
                case 283: excute_DIVR   (); return RT_OK;
                case 284: excute_SQRT   (); return RT_OK;
                case 285: excute_SIN    (); return RT_OK;
                case 286: excute_COS    (); return RT_OK;
                case 287: excute_TAN    (); return RT_OK;
                case 288: excute_LN     (); return RT_OK;
                case 289: excute_EXP    (); return RT_OK;
                case 290: excute_MOVR   (); return RT_OK;
                case 291: excute_DTR    (); return RT_OK;
                case 292: excute_ROUND  (); return RT_OK;
                case 293: excute_TRUNC  (); return RT_OK;
                }
        break;
        }

    return RT_CANCEL;
    }

void excute_ADDR(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) += *(float*)AP(0);
        }
    }

void excute_SUBR(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) -= *(float*)AP(0);
        }
    }

void excute_MULR(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) *= *(float*)AP(0);
        }
    }

void excute_DIVR(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) /= *(float*)AP(0);
```

```
        }
    }

void excute_SQRT(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = sqrtf(*(float*)AP(0));
        }
    }

void excute_SIN(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = sinf(*(float*)AP(0));
        }
    }

void excute_COS(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = cosf(*(float*)AP(0));
        }
    }

void excute_TAN(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = tanf(*(float*)AP(0));
        }
    }

void excute_LN(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = logf(*(float*)AP(0));
        }
    }

void excute_EXP(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = expf(*(float*)AP(0));
        }
    }

void excute_MOVR(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = *(float*)AP(0);
        }
    }
```

```
void excute_DTR(void) {
    if (theResState->itStackData & 1) {
        *(float*)AP(1) = (float)(*(int32_t*)AP(0));
        }
    }

void excute_ROUND(void) {
    if (theResState->itStackData & 1) {
        *(int32_t*)AP(1) = (int32_t)(*(float*)AP(0) + .5f);
        }
    }

void excute_TRUNC(void) {
    if (theResState->itStackData & 1) {
        *(int32_t*)AP(1) = (int32_t)(*(float*)AP(0));
        }
    }
```

由于添加浮点运算不涉及任何硬件操作，这里我们只需要响应指令派发消息 CUSTOM_LGC_INS_DISPATCH 即可。280 ~ 293 是这些指令对应的指令号，这些指令号通过 ManagerFun.xml 文件传递给 GUTTA Ladder Editor 软件。若在 GUTTA Ladder Editor 中编辑了这些指令，GUTTA Ladder Editor 最终会将这些指令号连同操作数一起下载到 EC30-EKSTM32 的 PLC 程序区。EC30-EKSTM32 的 PLC 解释系统若发现指令号大于或等于 224，则会尝试向用户二次开发程序发送 CUSTOM_LGC_INS_DISPATCH 消息。

在 CUSTOM_LGC_INS_DISPATCH 消息的处理函数中，我们根据指令号 param，分别调用对应的指令处理函数。由于这些指令都只涉及简单的数学运算，实现非常简单。需要特别感谢的是 IAR 自带的数学运算库。一般说来，标准的<math.h>数学库只需要实现双精度浮点运算（double）即可。IAR 提供的数学运算库，不但有标准的双精度浮点运算，还特别提供了精简版单精度浮点运算（float），这些函数需要在标准的函数名后加上“f”后缀。也正因为如此，二次开发的工程在编译后，体积不到 10K。下面是 link 信息文件的纪录。

```
  7 224 bytes of readonly  code memory
    16 bytes of readonly  data memory
    88 bytes of readwrite data memory

Errors: none
Warnings: none
```

测试

将项目提供的 **ManagerFun.xml** 文件替换掉 GUTTA Ladder Editor 软件安装目录下 \GuttaLad\EC30-EKSTM32**ManagerFun.xml** 文件。

打开软件 GUTTA Flash Utility。选择二次开发项目生成的程序文件 EC30-EKSTM32.hex，通过串口将这份程序下载到试验板 EC30-EKSTM32-EVAL。

打开项目自带的测试程序 ProjectForKeil-newlib-FloatSample.vcw，进入连线状态，通过改变每条指令的输入参数，测试每条指令是否工作正常。

相关下载

浮点运算二次开发项目下载：

ProjectForKeil-newlib-FloatSample.zip