# Reference to the Standard C, Math and ARS Library for the SHARC Compiler `g21k`

Dirk Bächle

Technische Universität Hamburg Harburg
Technische Informatik VI (Verteilte Systeme)
Prof. Dr. Georg-Friedrich Mayer-Lindenberg

December 8, 2003

**Abstract**

This is a short reference to the Standard C, Math and ARS Library for the SHARC *G21k* utils. It contains a list of all available functions like `asin` and `pow`. Each entry is provided with a description of what it does and its syntax. Unless otherwise noted, the usage of Standard C functions requires to include the header file `g21k/stdlib.h` by saying

```
#include <g21k/stdlib.h>
```

at the start of your C file. The same holds for the Math library and the `g21k/math.h` header or the ARS library and the `g21k/ars.h` header, respectively.
If you want to use math functions from `libm.a`, remember to specify the option

```
-lm
```

to the linker `ld21k` and the C compiler `g21k`. As default `double` values are treated like `float`s, so if you need true double-precision you have to add the

```
-fno-short-double
```

switch.
Likewise, the usage of the ARS library `libars.a` requires the option

```
-lars
```

to linker and C compiler.

# 1 The Standard C library libc.a

## abs

**Description**: Computes the absolute value of its integer input.

**Synopsis**: `int abs(int x);`

| | |
|---|---|
| `x` | Input integer |
| `Returns` | Absolute value of x |

## atexit

**Description**: This C language subroutine provides for function registration. These functions will be executed in the reverse order of registration at program exit.

**Synopsis**: `int atexit(void (*func)(void));`

| | |
|---|---|
| `func` | Pointer to the function |
| `Returns` | 1 if the function can not be registered, 0 if the function is registered. |

## atof

**Description**: Converts an ascii string to a floating point value.

**Synopsis**: `double atof(char *p);`

| | |
|---|---|
| `p` | Pointer to the string |
| `Returns` | Converted double value |

## atoff

**Description**: Converts an ascii string to a floating point value.

**Synopsis**: `float atoff(char *p);`

| | |
|---|---|
| `p` | Pointer to the string |
| `Returns` | Converted float value |

## atoi

**Description**: Converts an ascii string to an integer value.

**Synopsis**: `int atoi(char *p);`

| | |
|---|---|
| `p` | Pointer to the string |
| `Returns` | Converted integer value |

## atol

**Description**: Converts an ascii string to a long integer value.

**Synopsis**: `long int atol(char *p);`

| | |
|---|---|
| `p` | Pointer to the string |
| `Returns` | Converted long integer value |

## avg

**Description**: This C language subroutine computes the average value of the two inputs.

**Synopsis**: `int avg (int x, int y);`

| | |
|---|---|
| `x` | First value |
| `y` | Second value |
| `Returns` | Average of x and y |

**Remarks**: If you expect large integer values you should not use this function, because the assembler code uses the simple formula $avg = (x + y)/2$. Replace the function `avg` with $avg = x + (y - x)/2$ instead.

## bsearch

**Description**: This C language subroutine performs a binary search on an array of data.

**Synopsis**: `char *bsearch ((void *) key, (void *) base, unsigned nel, int keysize, int (*compare)();`

| | |
|---|---|
| `key` | Pointer to the key that is searched for |
| `base` | Pointer to initial member of the array |
| `nel` | Number of elements in the array |
| `keysize` | Size of an array member |
| `compare` | Pointer to compare function |
| `Returns` | 0 if no match, pointer to the key if a match is found |

## calloc

**Description**: Allocates a number of objects of size size on the heap.

**Synopsis**: `void *calloc(int nmemb, size_t size);`

| | |
|---|---|
| `nmemb` | Number of members to allocate |
| `size` | Number of bytes per member |
| `Returns` | Pointer to allocated memory, or NULL if allocation failed |

**See also**: free, malloc, realloc

## clip

**Description**: Computes the average value of the two inputs.

**Synopsis**: `int clip (int x, int y);`

| | |
|---|---|
| `x` | First integer value |
| `y` | Second integer value |
| `Returns` | Average of x and y |

## div

**Description**: Computes the integer quotient of its integer inputs.

**Synopsis**: `div_t div(int x, int y);`

| | |
|---|---|
| `x` | Integer input, numerator |
| `y` | Integer input, denominator |
| `Returns` | A div_t struct |

## exit

**Description**: Used to exit a C program. The return code is stored in R0.

**Synopsis**: `void exit(int return_code);`

| | |
|---|---|
| `return_code` | Return code |

## free

**Description**: Frees heap memory that was allocated by malloc, calloc or realloc.

**Synopsis**: `void free(void *p);`

`p` Pointer to memory that is freed

**See also**: calloc, malloc, realloc

# fmaxf

**Description**: Computes the maximum value of the two float inputs.

**Synopsis**: `float fmaxf(float x, float y);`

| | |
|---|---|
| `x` | First float value |
| `y` | Second float value |
| `Returns` | max(x,y) |

# fminf

**Description**: Computes the minimum value of the two float inputs.

**Synopsis**: `float fminf(float x, float y);`

| | |
|---|---|
| `x` | First float value |
| `y` | Second float value |
| `Returns` | min(x,y) |

# fsign

**Description**: Computes the copysign value of the two inputs.

**Synopsis**: `double copysign(double x, double y);`

| | |
|---|---|
| `x` | First double input |
| `y` | Second double input |
| `Returns` | Copysign value of x and y |

# fsignf

**Description**: Computes the copysign value of the two inputs.

**Synopsis**: `float copysign(float x, float y);`

| | |
|---|---|
| `x` | First float input |
| `y` | Second float input |
| `Returns` | Copysign value of x and y |

## getenv

**Description**: Provides a hook for the getenv routine of the library. Since there is no defined environment, this always returns a NULL.

**Synopsis**: `char *getenv(const char *name);`

| | |
|---|---|
| `name` | Environment variable |
| `Returns` | NULL |

## idle

**Description**: Executes the ADSP-21020 idle instruction. This function does not return!

**Synopsis**: `void idle(void);`

## interrupt

**Description**: Controls the interrupts received by a program.

**Header**: `#include <signal.h>`

**Synopsis**: `void (*interrupt(int sig, void (*func)(int))) (int);`

| | |
|---|---|
| `sig` | Interrupt number |
| `func` | Pointer to interrupt handler function |

**See also**: raise

## isalnum

**Description**: Returns non-zero if the input is alpha-numeric, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isalnum(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is not alpha-numeric |

## isalpha

**Description**: Returns non-zero if the input is alphabetic, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isalpha(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is not alphabetic |

## iscntrl

**Description**: Returns non-zero if the input is a control character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int iscntrl(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is no control character |

## isdigit

**Description**: Returns non-zero if the input is a digit, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isdigit(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is no digit |

## isgraph

**Description**: Returns non-zero if the input is a graphics character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isgraph(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is no graphics character |

## islower

**Description**: Returns non-zero if the input is a lower case character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int islower(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is not lower case |

## isprint

**Description**: Returns non-zero if the input is a printable character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isprint(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is not printable |

## ispunct

**Description**: Returns non-zero if the input is a punctuation character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int ispunct(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if c is no punctuation character |

## isspace

**Description**: Returns non-zero if the input is a space character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isspace(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if c is no space character |

## isupper

**Description**: Returns non-zero if the input is a upper case character, zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isupper(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is not upper case |

## isxdigit

**Description**: Returns non-zero if the input is a hex digit (0–9, A–F, a–f), zero otherwise.

**Header**: `#include <ctype.h>`

**Synopsis**: `int isxdigit(int c);`

| | |
|---|---|
| `c` | Character code |
| `Returns` | "0" if character is no hex digit |

## labs

**Description**: Computes the absolute value of its long integer input.

**Synopsis**: `long int labs(long int x);`

| | |
|---|---|
| `x` | Input long integer |
| `Returns` | Absolute value of x |

## ldiv

**Description**: Computes the integer quotient of its long integer inputs.

**Synopsis**: `ldiv_t ldiv(long x, long y);`

| | |
|---|---|
| `x` | Long input, numerator |
| `y` | Long input, denominator |
| `Returns` | A ldiv_t struct |

## lmax

**Description**: Computes the maximum value of the two integer inputs.

**Synopsis**: `long int max(long int x, long int y);`

| | |
|---|---|
| `x` | First integer value |
| `y` | Second integer value |
| `Returns` | max(x,y) |

## lmin

**Description**: Computes the minimum value of the two long integer inputs.

**Synopsis**: `long int min(long int x, long int y);`

| | |
|---|---|
| `x` | First integer value |
| `y` | Second integer value |
| `Returns` | min(x,y) |

## malloc

**Description**: Allocates size bytes on the heap.

**Synopsis**: `void *malloc(size_t size);`

| | |
|---|---|
| `size` | Number of bytes to allocate |
| `Returns` | Pointer to allocated memory, or NULL if allocation failed |

**See also**: free, calloc, realloc

## max

**Description**: Computes the maximum value of the two integer inputs.

**Synopsis**: `int max(int x, int y);`

| | |
|---|---|
| `x` | First integer value |
| `y` | Second integer value |
| `Returns` | max(x,y) |

## memchr

**Description**: Searches for a character within a range of memory.

**Header**: `#include <string.h>`

**Synopsis**: `void *memchr(const void *s1, int c, size_t n);`

| | |
|---|---|
| `s1` | Pointer to start in memory |
| `c` | Character to search for |
| `n` | Size of memory to search |
| `Returns` | Pointer to found char, or NULL if char was not found |

## memcmp

**Description**: Compares two ranges of memory.

**Header**: `#include <string.h>`

**Synopsis**: `int memcmp(const void *s1, const void *s2, size_t size);`

| | |
|---|---|
| `s1` | Pointer to start in memory 1 |
| `s2` | Pointer to start in memory 2 |
| `size` | Number of bytes to compare |
| `Returns` | "0" if both memory areas are equal |

## memcpy

**Description**: Copies one range of memory into the other.

**Header**: `#include <string.h>`

**Synopsis**: `void *memcpy(const void *s1, const void *s2, size_t size);`

| | |
|---|---|
| `s1` | Pointer to start in destination memory |
| `s2` | Pointer to start in source memory |
| `size` | Number of bytes to copy |
| `Returns` | Pointer to start of destination memory |

## memmove

**Description**: Copies one range of memory into the other without overlap problems.

**Header**: `#include <string.h>`

**Synopsis**: `void *memcpy(const void *s1, const void *s2, size_t size);`

| | |
|---|---|
| `s1` | Pointer to start in destination memory |
| `s2` | Pointer to start in source memory |
| `size` | Number of bytes to move |
| `Returns` | Pointer to start of destination memory |

## memset

**Description**: Sets a range of memory to a specific value.

**Header**: `#include <string.h>`

**Synopsis**: `void *memset(void *s1, int c, size_t size);`

| | |
|---|---|
| `s1` | Pointer to start of memory |
| `c` | Character for initializing the memory |
| `size` | Number of bytes to set |
| `Returns` | Pointer to start of memory |

## min

**Description**: Computes the minimum value of the two integer inputs.

**Synopsis**: `int min(int x, int y);`

| | |
|---|---|
| `x` | First integer value |
| `y` | Second integer value |
| `Returns` | min(x,y) |

## poll_flag_in

**Description**: Provides an interface to the input flags of the ADSP-21020.

**Header**: `#include <21020.h>`

**Synopsis**: `int poll_flag_in(int flag, int mode);`

| | |
|---|---|
| `flag` | Flag number (0–3) |
| `mode` | Flag operation:<br>0 = transition LOW to HIGH<br>1 = transition HIGH to LOW<br>2 = flag HIGH<br>3 = flag LOW<br>4 = transition<br>other = read only flag |
| `Returns` | "0" for modes 0 and 4, flag value (1 or 0) for any other modes. An invalid flag results in a "-1". |

## qsort

**Description**: This C language subroutine is for quick sort of an array. The algorithm sorts an array base[0]...base[n-1] of objects of size size into ascending order. The array is overwritten with the sorted elements.

**Synopsis**: `void qsort (void *base, size_t n, size_t size, int (*compare)(const void *, const void *));`

| | |
|---|---|
| `base` | Pointer to the array |
| `n` | Number of elements in the array |
| `size` | Size of an array member |
| `compare` | Pointer to compare function |

## raise

**Description**: Sends the signal to the program by setting a corresponding bit in IRPTL register.

**Header**: `#include <signal.h>`

**Synopsis**: `int raise(int sig);`

| | |
|---|---|
| `sig` | Interrupt number |
| `Returns` | "0" if successful, non-zero else |

**See also**: interrupt

## rand

**Description**: Returns a pseudo-random number between 0 and RAND_MAX

(= 2147483647 as defined in `stdlib.h`).

**Synopsis**: `int rand(void);`

| | |
|---|---|
| `Returns` | Pseudo-random number |

**See also**: srand

## realloc

**Description**: Reallocates a piece of memory.

**Synopsis**: `void *realloc(void *ptr, size_t new_size);`

| | |
|---|---|
| `ptr` | Pointer to current memory area |
| `new_size` | New size of the memory area |
| `Returns` | Pointer to new memory area, NULL if allocation failed |

**See also**: free, malloc, calloc

## set_flag

**Description**: Provides an interface to the input flags of the ADSP-21020.

**Header**: `#include <21020.h>`

**Synopsis**: `int set_flag(int flag, int mode);`

| | |
|---|---|
| `flag` | Flag number (0–3) |
| `mode` | Flag operation (0 = set, 1 = clr, 2 = toggle, 3 = read) |
| `Returns` | "0" for modes 0 and 4, flag value (1 or 0) for any other modes. An invalid flag results in a "-1". |

## srand

**Description**: Sets the seed value, used for generating a pseudo-random number by rand().

**Synopsis**: `void srand(int seed);`

| | |
|---|---|
| `seed` | New seed value |

**See also**: rand

## sign

**Description**: Computes the copysign value of the two inputs.

**Synopsis**: `int copysign(int x, int y);`

| | |
|---|---|
| `x` | First integer input |
| `y` | Second integer input |
| `Returns` | Copysign value of x and y |

## signal

**Description**: Controls the interrupts received by a program.

**Header**: `#include <signal.h>`

**Synopsis**: `void (*signal(int sig, void (*func)(int))) (int);`

| | |
|---|---|
| `sig` | Interrupt number |
| `func` | Pointer to interrupt handler function |

**See also**: interrupt, raise

## strcat

**Description**: Concatenates one string to another.

**Header**: `#include <string.h>`

**Synopsis**: `char *strcat(char *s1, const char *s2);`

| | |
|---|---|
| `s1` | Pointer to string that gets extended |
| `s2` | Pointer to string that is appended |
| `Returns` | Pointer to s1 |

## strchr

**Description**: Searches for a character within a string.

**Header**: `#include <string.h>`

**Synopsis**: `char *strchr(const char *s1, int c);`

| | |
|---|---|
| `s1` | Pointer to string |
| `c` | Character that is searched for |
| `Returns` | Pointer to found character,<br>or NULL if search failed |

## strcmp

**Description**: Compares two strings.

**Header**: `#include <string.h>`

**Synopsis**: `int strcmp(const char *s1, const char *s2);`

| | |
|---|---|
| `s1` | Pointer to first string |
| `s2` | Pointer to second string |
| `Returns` | 0, if s1 equals s2<br>a negative number, if s1 < s2<br>a positive number, if s1 > s2 |

## strcpy

**Description**: Copies one string into another.

**Header**: `#include <string.h>`

**Synopsis**: `char *strcpy(char *s1, const char *s2);`

| | |
|---|---|
| `s1` | Pointer to destination string |
| `s2` | Pointer to source string |
| `Returns` | Pointer to s1 |

## strcspn

**Description**: Computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters not from the string pointed to by s2.

**Header**: `#include <string.h>`

**Synopsis**: `size_t strcspn(const char *s1, const char *s2);`

| | |
|---|---|
| `s1` | String that is scanned |
| `s2` | Array of characters for the check |
| `Returns` | Number of first characters of s1 that are<br>**not** contained in s2 |

## strlen

**Description**: Determines the length of a string.

**Header**: `#include <string.h>`

**Synopsis**: `int strlen(const char *s1);`

| | |
|---|---|
| `s1` | Pointer to string |
| `Returns` | Length of s1 |

## strncat

**Description**: Concatenates one string to another, copying at most N characters.

**Header**: `#include <string.h>`

**Synopsis**: `char *strncat(char *s1, const char *s2, size_t size);`

| | |
|---|---|
| `s1` | Pointer to destination string |
| `s2` | Pointer to source string |
| `size` | Maximum number of characters to copy |
| `Returns` | Pointer to s1 |

## strncmp

**Description**: Compares two strings with a limit on the length.

**Header**: `#include <string.h>`

**Synopsis**: `int strncmp(const char *s1, const char *s2, size_t size);`

| | |
|---|---|
| `s1` | Pointer to destination string |
| `s2` | Pointer to source string |
| `size` | Maximum number of characters to copy |
| `Returns` | 0, if s1 equals s2 |
| | a negative number, if s1 < s2 |
| | a positive number, if s1 > s2 |

## strncpy

**Description**: Copies at most N characters from one string to the other.

**Header**: `#include <string.h>`

**Synopsis**: `char *strncpy(char *s1, const char *s2, size_t size);`

| | |
|---|---|
| `s1` | Pointer to destination string |
| `s2` | Pointer to source string |
| `size` | Maximum number of characters to copy |
| `Returns` | Pointer to s1 |

## strpbrk

**Description**: Returns a pointer to the first occurance of a character from s2 found in s1.

**Header**: `#include <string.h>`

**Synopsis**: `size_t strpbrk(const char *s1, const char *s2);`

| | |
|---|---|
| `s1` | String that is scanned |
| `s2` | Array of characters for the check |
| `Returns` | Pointer to first occurance of a character from s2 in s1 |

## strspn

**Description**: Computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters from the string pointed to by s2.

**Header**: `#include <string.h>`

**Synopsis**: `size_t strspn(const char *s1, const char *s2);`

| | |
|---|---|
| `s1` | String that is scanned |
| `s2` | Array of characters for the check |
| `Returns` | Number of first characters of s1 that are contained in s2 |

## strstr

**Description**: Locates the first occurance of the string s2 that is located in the string s1. (Not including NULL)

**Header**: `#include <string.h>`

**Synopsis**: `int strstr(const char *s1, const char *s2);`

| | |
|---|---|
| `s1` | String that is scanned |
| `s2` | Substring that is searched for in s1 |
| `Returns` | Position of the first occurance of s2 in s1 |

## strtod

**Description**: Converts an ascii string to a floating point value.

**Synopsis**: `double strtod(char *p, char **ptr);`

| | |
|---|---|
| `p` | Pointer to the string |
| `ptr` | If ptr is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by ptr |
| `Returns` | Converted double value |

## strtodf

**Description**: Converts an ascii string to a floating point value.

**Synopsis**: `float strtodf(char *p, char **ptr);`

| | |
|---|---|
| `p` | Pointer to the string |
| `ptr` | If ptr is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by ptr |
| `Returns` | Converted float value |

## strtol

**Description**: Converts an ascii string to an integer value, according to the given base.

**Synopsis**: `int strtol(char *p, char **ptr, int base);`

| | |
|---|---|
| `p` | Pointer to the string |
| `ptr` | If ptr is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by ptr |
| `base` | The base of the given number (2–36), if 0 the function assumes a base of: 16 if p starts with '0x' 8 if p starts with '0' 10 for none of the above |
| `Returns` | Converted integer value |

## strtoul

**Description**: Converts an ascii string to a long integer value, according to the given base.

**Synopsis**: `unsigned long int strtoull(char *p, char **ptr, int base);`

| | |
|---|---|
| `p` | Pointer to the string |
| `ptr` | If ptr is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by ptr |
| `base` | The base of the given number (2–36), if 0 the function assumes a base of: 16 if p starts with '0x' 8 if p starts with '0' 10 for none of the above |
| `Returns` | Converted long integer value |

## strtok

**Description**: Breaks an input string into tokens separated by a string of separators.

**Synopsis**: `char *strtok(char *s1, const char *s2);`

| | |
|---|---|
| `s1` | Pointer to string that is separated |
| `s2` | Pointer to array of separator characters |
| `Returns` | Pointer to the next token, or NULL if no token was found |

## tolower

**Description**: Converts an upper case letter to lowercase.

**Header**: `#include <ctype.h>`

**Synopsis**: `int tolower(int c);`

| | |
|---|---|
| `c` | Input character |
| `Returns` | Lowercase of c |

## toupper

**Description**: Converts a lower case letter to uppercase.

**Header**: `#include <ctype.h>`

**Synopsis**: `int toupper(int c);`

| | |
|---|---|
| `c` | Input character |
| `Returns` | Uppercase of c |

# 2 The Math library libm.a

## acos

**Description**: Computes the ArcCos of its floating point input.

**Synopsis**: `double acos(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | ArcCos of x |

## acosf

**Description**: Computes the ArcCos of its floating point input.

**Synopsis**: `float acos(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | ArcCos of x |

## asin

**Description**: Computes the ArcSine of its floating point input.

**Synopsis**: `double asin(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | ArcSine of x |

## asinf

**Description**: Computes the ArcSine of its floating point input.

**Synopsis**: `float asinf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | ArcSine of x |

## atan

**Description**: Computes the ArcTangens of its floating point input.

**Synopsis**: `double atan(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | ArcTangens of x |

## atanf

**Description**: Computes the ArcTangens of its floating point input.

**Synopsis**: `float atanf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | ArcTangens of x |

## atan2

**Description**: Computes the ArcTangens for x/y.

**Synopsis**: `double atan2(double x, double y);`

| | |
|---|---|
| `x` | Double input, nominator |
| `y` | Double input, denominator |
| `Returns` | ArcTangens of x/y |

## atan2f

**Description**: Computes the ArcTangens for x/y.

**Synopsis**: `float atan2f(float x, float y);`

| | |
|---|---|
| `x` | Float input, nominator |
| `y` | Float input, denominator |
| `Returns` | ArcTangens of x/y |

## ceil

**Description**: Computes the value of the smallest integer greater then or equal to the input.

**Synopsis**: `double ceil(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Ceil of x |

## ceilf

**Description**: Computes the value of the smallest integer greater then or equal to the float input.

**Synopsis**: `float ceilf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Ceil of x |

## cos

**Description**: Computes the Cosine of its floating point input.

**Synopsis**: `double cos(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Cosine of x |

## cosf

**Description**: Computes the Cosine of its floating point input.

**Synopsis**: `float cosf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Cosine of x |

## cosh

**Description**: Computes the Hyperbolic Cosine of its floating point input.

**Synopsis**: `double cosh(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Hyperbolic Cosine of x |

## coshf

**Description**: Computes the Hyperbolic Cosine of its floating point input.

**Synopsis**: `float coshf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Hyperbolic Cosine of x |

## exp

**Description**: Computes the exponential value of its floating point input.

**Synopsis**: `double exp(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | $e^x$ |

## expf

**Description**: Computes the exponential value of its floating point input.

**Synopsis**: `float expf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | $e^x$ |

## fabs

**Description**: Computes the floating point absolute value of its input.

**Synopsis**: `double fabs(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Absolute value of x |

## fabsf

**Description**: Computes the floating point absolute value of its input.

**Synopsis**: `float fabsf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Absolute value of x |

## floor

**Description**: Computes the greatest integer less than or equal to the input.

**Synopsis**: `double floor(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Floor of x |

## floorf

**Description**: Computes the greatest integer less than or equal to the input.

**Synopsis**: `float floorf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Floor of x |

## fmod

**Description**: Computes the floating point modulo of its two floating point inputs.

**Synopsis**: `double fmod(double x, double y);`

| | |
|---|---|
| `x` | Double input |
| `y` | Double input |
| `Returns` | x modulo y |

## fmodf

**Description**: Computes the floating point modulo of its two floating point inputs.

**Synopsis**: `float fmodf(float x, float y);`

| | |
|---|---|
| `x` | Float input |
| `y` | Float input |
| `Returns` | x modulo y |

## frexp

**Description**: Seperates the mantissa of a floating point number from its exponent.

**Synopsis**: `double frexp(double x, int *exp);`

| | |
|---|---|
| `x` | Double input |
| `exp` | Pointer to exponent |
| `Returns` | Separated mantissa |

## frexpf

**Description**: Seperates the mantissa of a floating point number from its exponent.

**Synopsis**: `float frexpf(float x, int *exp);`

| | |
|---|---|
| `x` | Float input |
| `exp` | Pointer to exponent |
| `Returns` | Separated mantissa |

## isinf

**Description**: Returns a non-zero value if its double-precision floating point input value is infinite.

**Synopsis**: `int isinf(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | 0 if x is finite |

## isinff

**Description**: Returns a non-zero value if its single-precision floating point input value is infinite.

**Synopsis**: `int isinff(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | 0 if x is finite |

## isnan

**Description**: Returns a non-zero value if its double-precision floating point input value is NaN (Not a Number).

**Synopsis**: `int isnan(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | 0 if x is a valid number |

## isnanf

**Description**: Returns a non-zero value if its single-precision floating point input value is NaN (Not a Number).

**Synopsis**: `int isnanf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | 0 if x is a valid number |

## ldexp

**Description**: Scales a floating point number by a factor of $2^{exp}$

**Synopsis**: `double ldexp(double x, int exp);`

| | |
|---|---|
| `x` | Double input |
| `exp` | Integer input, exponent |
| `Returns` | $x \cdot 2^{exp}$ |

## ldexpf

**Description**: Scales a floating point number by a factor of $2^{exp}$

**Synopsis**: `float ldexpf(float x, int exp);`

| | |
|---|---|
| `x` | Float input |
| `exp` | Integer input, exponent |
| `Returns` | $x \cdot 2^{exp}$ |

## log

**Description**: Computes the natural logarithm of the floating point input.

**Synopsis**: `double log(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | $\log_e(x)$ |

## log10

**Description**: Computes the base-10 logarithm of the floating point input.

**Synopsis**: `double log10(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | $\log_{10}(x)$ |

## logf

**Description**: Computes the natural logarithm of the floating point input.

**Synopsis**: `float logf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | $\log_e(x)$ |

## log10f

**Description**: Computes the base-10 logarithm of the floating point input.

**Synopsis**: `float log10f(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | $\log_{10}(x)$ |

## modf

**Description**: Seperates the integer part of a floating point number from its fractional part.

**Synopsis**: `double modf(double x, double *iptr);`

| | |
|---|---|
| `x` | Double input |
| `iptr` | Pointer to integer part of x |
| `Returns` | Fractional part of x |

## modff

**Description**: Seperates the integer part of a floating point number from its fractional part.

**Synopsis**: `float modff(float x, float *iptr);`

| | |
|---|---|
| `x` | Double input |
| `iptr` | Pointer to integer part of x |
| `Returns` | Fractional part of x |

## pow

**Description**: Computes the value of $x^y$.

**Synopsis**: `double pow(double x, double y);`

| | |
|---|---|
| `x` | Double input, base |
| `y` | Double input, exponent |
| `Returns` | $x^y$ |

## powf

**Description**: Computes the value of $x^y$.

**Synopsis**: `float powf(float x, float y);`

| | |
|---|---|
| `x` | Float input, base |
| `y` | Float input, exponent |
| `Returns` | $x^y$ |

## sin

**Description**: Computes the Sine of its floating point input.

**Synopsis**: `double sin(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Sine of x |

## sinf

**Description**: Computes the Sine of its floating point input.

**Synopsis**: `float sinf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Sine of x |

## sinh

**Description**: Computes the Hyperbolic Sine of its floating point input.

**Synopsis**: `double sinh(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Hyperbolic Sine of x |

## sinhf

**Description**: Computes the Hyperbolic Sine of its floating point input.

**Synopsis**: `float sinhf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Hyperbolic Sine of x |

## sqrt

**Description**: Computes the square root of its floating point input.

**Synopsis**: `double sqrt(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | $\sqrt{x}$ |

## sqrtf

**Description**: Computes the square root of its floating point input.

**Synopsis**: `float sqrtf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | $\sqrt{x}$ |

## tan

**Description**: Computes the Tangens of its floating point input.

**Synopsis**: `double tan(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Tangens of x |

## tanf

**Description**: Computes the Tangens of its floating point input.

**Synopsis**: `float tanf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Tangens of x |

## tanh

**Description**: Computes the Hyperbolic Tangens of its floating point input.

**Synopsis**: `double tanh(double x);`

| | |
|---|---|
| `x` | Double input |
| `Returns` | Hyperbolic Tangens of x |

# tanhf

**Description**: Computes the Hyperbolic Tangens of its floating point input.

**Synopsis**: `float tanhf(float x);`

| | |
|---|---|
| `x` | Float input |
| `Returns` | Hyperbolic Tangens of x |

# 3 The ARS library `libars.a`

## ars_barrier

**Description**: Similar to the PVM barrier, this function can be used to synchronize all SHARCs of a cluster to a point in their control flow.

**Synopsis**: `void ars_barrier();`

**Remarks**: This function assumes that a matching `ars_barrier` is called at all other SHARC processors.

## ars_get_id

**Description**: Gets the multiprocessing ID of the current processor.

**Synopsis**: `int ars_get_id(void);`

`Returns` The multiprocessing ID (1–4)

## ars_signal

**Description**: Sends a signal to the specified remote processor and waits for an acknowledge.

**Synopsis**: `void ars_signal(int ID);`

`ID` ID of the remote processor (1–4)

**Remarks**: This function assumes that a matching `ars_wait` is called at the remote processor.

## ars_wait

**Description**: Waits for a signal from the specified remote processor and sends an acknowledge.

**Synopsis**: `void ars_wait(int ID);`

`ID` ID of the remote processor (1–4)

**Remarks**: This function assumes that a matching `ars_signal` is called at the remote processor.

## idma_read

**Description**: Reads a 16-bit value from the 2181 memory location via IDMA.

**Synopsis**: `int idma_read(int address);`

| | |
|---|---|
| `address` | Address to the memory of the 2181 |
| `Returns` | 16-bit value at the address |

## idma_read_float

**Description**: Reads a 32-bit float value (single precision) from the 2181 memory locations (address) and (address+1) via IDMA.

**Synopsis**: `float idma_read_float(int address);`

| | |
|---|---|
| `address` | Address to the memory of the 2181 |
| `Returns` | 32-bit float value at the given address |

**Remarks**: Please, note that this function implicitly contains a `pack_float`, i.e. the byte order of the float value is changed!

## idma_write

**Description**: Writes a 16-bit value to the 2181 memory location via IDMA.

**Synopsis**: `void idma_write(int address, int value);`

| | |
|---|---|
| `address` | Address to the memory of the 2181 |
| `value` | 16-bit value |

## idma_write_float

**Description**: Writes a 32-bit float value (single precision) to the 2181 memory locations (address) and (address+1) via IDMA.

**Synopsis**: `void idma_write_float(int address, float value);`

| | |
|---|---|
| `address` | Address to the memory of the 2181 |
| `value` | 32-bit float value |

**Remarks**: Please, note that this function implicitly contains a `pack_float`, i.e. the byte order of the float value is changed!

## led_off

**Description**: Switches off the red LED for the current processor.

**Synopsis**: `void led_off();`

## led_on

**Description**: Switches on the red LED for the current processor.

**Synopsis**: `void led_on();`

## pack_float

**Description**: Changes the byte order of the given 32-bit value from 1234 to 4321, i.e. reverses it. Can be used to convert float values between Linux and the SHARCs regarding their endianness.

**Synopsis**: `int pack_float(int value);`

| | |
|---|---|
| `value` | 32-bit word |
| `Returns` | 32-bit word with reversed byte order |

# Index