

HC11 Specific Functions and Header Files

These functions provide access to HC11 specific features. They are part of `libc.a` so you do not need to do anything special to use them. You must include the header files `hc11.h` to reference the functions.

The header file `hc11.h` defines the standard IO registers. If you move the IO base address or use a different processor, you may need to change some of the definitions. It also defines the `BaudRate` enum based on a 8 MHz clock. If your system uses a different clock frequency, you must change the file and recompile `serial.c`. Lastly, it defines function prototypes and defines that are useful for accessing the peripheral registers.

Functions

`void setbaud(BaudRate)` sets the baud rate for the SCI port and enables both the transmitter and receiver functions.

`unsigned int read_sci()` reads a 9-bit "byte" from the SCI port and returns it. Use `getchar()` if you want a 8-bit byte.

`unsigned char read_spi()` reads a character from the SPI port. The SPI functions must have been initialized.

void **write_eeprom**(unsigned char *addr, unsigned char c) writes a byte to the internal EEPROM. Note that you cannot write to an internal EEPROM cell with code that is running out of the internal EEPROM.

void **write_sci**(unsigned int) writes a 9-bit “byte” to the SCI port. Use **putchar()** if you are writing only 8 bits of data.

void **write_spi**(unsigned char) writes a byte to the SPI port. The SPI functions must have been initialized.

Standard C Library and Header Files

The compiler comes with a subset of the Standard C library. Most functions are Standard C conformant, with the most notable exception being that `printf()` only supports a subset of the format characters since supporting the full set would make the code too large.

Assert Macro: ASSERT.H

The macro `assert(test)` aborts the execution of the program if the test condition evaluates to 0.

Character Classification: CTYPE.H

The following functions categorize input according to the ASCII character set:

`int isalnum(int c)` returns nonzero if `c` is a digit or alphabetic.

`int isalpha(int c)` returns nonzero if `c` is an alphabetic.

`int iscntrl(int c)` returns nonzero if `c` is a control character (e.g. FF, BELL, LF ..etc.).

`int isdigit(int c)` returns nonzero if `c` is a digit.

`int isgraph(int c)` returns nonzero if `c` is a printable character and not *space*.

`int islower(int c)` returns nonzero if `c` is a lower case alphabetic.

`int isprint(int c)` returns nonzero if `c` is a printable character.

`int ispunct(int c)` returns nonzero if `c` is a printable character and is not space or a digit or an alphabetic.

`int isspace(int c)` returns nonzero if `c` is a space character including *space*, CR, FF, HT, NL, and VT.

`int isupper(int c)` returns nonzero if `c` is an upper case alphabetic.

`int isxdigit(int c)` returns nonzero if `c` is a hexadecimal digit.

`int tolower(int c)` returns the lower case version of `c` if `c` is an upper case character. Otherwise it returns `c`.

`int toupper(int c)` returns the upper case version of `c` if `c` is a lower case character. Otherwise it returns `c`.

Floating Point Characteristics: FLOAT.H

This file defines the floating point characteristics.

Implementation Limits: LIMITS.H

This header file defines the following implementation specific macros:

CHAR-BIT 8
CHAR-MAX 127
CHAR_MIN -128
INT_MAX 32767
INT_MIN -32768
LONG-MAX 2 1474836471,
LONG_MIN (-2147483647L- 1)
ULONG_MAX 4294967295UL
MB-LEN-MAX 1
SCHAR_MAX 127
SCHAR_MIN -128

SHRT_MAX 32767
SHRT_MIN -32768
UCHAR_MAX 255
UINT_MAX 65535
USHRT_MAX 65535

Floating Point

Functions: MATH.H

This file declares the floating point math routines:

double exp(double x) returns e^x to the x power.

double fabs(double x) returns the absolute value of x

double fmod(double x, double y) returns the remainder of x / y .

double log(double x) returns the natural logarithm of x.

double log10(double x) returns the base-10 logarithm of x.

double pow(double x, double y) returns x raised to the power y.

double sqrt(double x) returns the square root of x.

double sin(double x) returns the sine of x for x in radians.

double cos(double x) returns the cosine of x for x in radians.

double tan(double x) returns the tangent of x for x in radians.

double asin(double x) returns the arcsine of x for x in radians.

double acos(double x) returns the arccosine of x for x in radians.

double atan(double x) returns the arctangent of x for x in radians.

Setjmp: SET JMP.H

This header file defines the pseudo-type jmp_buf.

int setjmp(jmp_buf buffer) returns 0 when first called, and returns nonzero when "returned" from a subsequent longjmp call.

`void longjmp(jmp_buf buffer, int retval)` returns to the `setjmp()` return point with the value `retval`. If `retval` is 0, then 1 is returned. A `longjmp` can only be called if the `setjmp` point it is jumping to is still in the execution context.

Variable Argument Support: **STDARG.H**

`stdarg . h` provides support for variable argument processing. it defines the pseudo-type `va_list`, and three macros:

`va_start(va_list foo, <last-arg>)`, which initializes the variable "foo."

`va_arg(va_list foo, <promoted type>)`, which accesses the next argument. cast to the **specified type**.

`va_end(va_list foo)`, which ends the variable argument processing.

For example, `printf()` may be implemented using `vfprintf()` as follows:

```
#include <stdarg.h>
int printf(char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    vfprintf(fmt, ap);
    va_end(ap);
}
```

Standard Defines: **STDDEF.H**

`stddef . h` defines the macros `NULL` and `offsetof`. and the typedefed types `ptrdiff_t`, `size_t`, and `wchar_t`.

`offsetof(<structure type>, <member name>)` returns the byte offset of a structure member.

Standard Input Output: **STDIO.H**

Since standard file IO is not meaningful for an embedded microcontroller, much of the standard `stdio . h` content is not applicable. Nevertheless, some output functions are supported. You will need to initialize the SCI port before you use these functions (by calling the `setbaud()` function unless you are running under a monitor that already initialized the SCI port).

`int getchar()` returns a character from the SCI port using polled mode.

`int printf(char *fmt, ...)` prints out formatted text according to the format specifiers in the `fmt` string. The format specifiers are a subset of the standard formats:

- `%d` - prints the next argument as a decimal integer
- `%o` - prints the next argument as an unsigned octal integer
- `%x` - prints the next argument as an unsigned hexadecimal integer
- `%u` - prints the next argument as an unsigned decimal integer
- `%s` - prints the next argument as a C nul terminated string
- `%c` - prints the next argument as an ASCII character
- `%f` - prints the next argument as a floating point number (you must include the library `libfp.a` to use this)

If a `#` character is specified between `%o` and `%x`, then a leading 0 or 0x is printed respectively. If you specify `l` (letter `l`) between `%` and one of the integer format characters, then the argument is taken to be long, instead of `int`.

`int putchar(int c)` prints out 1 character. `Printf()` relies on this function to do its output. The library routine uses the SCI in polled mode to output a single character. You may need to replace this routine for your system. For convenience, output a `'\n'` newline character will cause a `'\r'` carriage return character to be output first.

`int puts(char *s)` prints out a string followed by NL. It uses the `putchar()` function.

`int sprintf(char *buf, char *fmt)` prints a formatted text into `buf` according to the format specifiers in `fmt`. The format specifiers are the same as in `printf()`.

In addition, to ease porting programs from other environments, `stdout` and `stderr` are defined as 0, and `FILE` is typedefed as `void`, and `fprintf(FILE *, char *fmt, ...)` is the same as `printf(char *fmt, ...)`.

Standard Library Functions: **STDLIB.H**

`stdlib.h` defines the macros `NULL` and `RAND_MAX` and typedefs `size_t`. It declares the following functions:

`int abs(int i)` returns the absolute value of `i`.

`int atoi(char *s)` converts the initial characters in `s` into an integer, or returns 0 if an error occurs.

`double atof(const char *s)` converts the initial characters in `s` into a double and returns it.

`long atol(char*s)` converts the initial characters in `s` into a long integer, or returns 0 if an error occurs.

`void *calloc(size_t nelem, size_t size)` returns a memory chunk large enough to hold `nelem` number of objects, each of size "size." The memory is initialized to zeroes. It allocates memory from the heap (i.e., you must call `_NewHeap()` before this or any of the memory allocation routines is called) and returns 0 if it cannot honor the request.

`void exit(status)` terminates the program. The exit value is written to the D register.

`void free(void *ptr)` frees a previously allocated heap memory.

`void *malloc(size_t size)` allocates a memory chunk of size "size" from the heap. It returns 0 if it cannot honor the request.

`int rand(void)` returns a pseudorandom number between 0 and RAND-MAX.

`void *realloc(void*ptr,size_tsize)` reallocates a previously allocated memory chunk with a new size.

`void srand(unsigned seed)` initializes the seed value for subsequent `rand()` calls.

`long strtol(char*s, char **endptr, int base)` converts the initial characters in `s` to a long integer according to the base. If `base` is 0, then `strtol` chooses the base depending on the initial characters (after the optional minus sign, if any) in `s`: `0x` or `0X` indicates a hexadecimal integer. `0` indicates an octal integer, with a decimal integer assumed otherwise. If `endptr` is not NULL, then `*endptr` will be set to where the conversion ends in `s`.

`unsigned long strtoul(char*s, char **endptr, int base)` is the same thing as `strtol()` except that the number to be converted is an unsigned long and the return value is unsigned long.

String Functions: STRING.H

string. h defines NULL and typedefs size_t, and the following string and character array functions:

void *memchr(void*s, int c, size_t n) searches for the first occurrence of c in the array s of size n. It returns the address of the matching element or the null pointer if no match is found.

int memcmp(void *s1, void *s2, size_t n) compares two arrays. each of size n. It returns 0 if the arrays are equal and greater than 0 if the first differed element in s1 is greater than the corresponding element in s2. Otherwise it returns a number less than 0.

void *memcpy(void *s1, void *s2, size_t n) copies s2 into s1, each of size n. The routine works correctly even if the inputs overlap. It returns s1.

void *memset(void *s, int c, size_t n) stores c in all elements of the array s of size n. It returns s.

char *strcat(char*s1, char*s2) concatenates s2 into s1. It returns s1.

char *strchr(char*s, int c) searches for the first occurrence of c in s, including its terminating null character. It returns the address of the matching element or the null pointer if no match is found.

int strcmp(char*s1, char*s2) compares two strings. It returns 0 if the strings are equal, and greater than 0 if the first differed element in s1 is greater than the corresponding element in s2. Otherwise, it returns a number less than 0.

char *strcpy(char*s1, char*s2) copies s2 into s1. It returns s1.

size_t strcspn(char*s1, char*s2) searches for the first element in s1 that matches any of the elements in s2. The terminating nulls are considered part of the strings. It returns the index where the match is found.

size_t strlen(char*s) returns the length of s.

char *strncat(char*s1, char*s2, size_t n) concatenates up to n elements, not including the terminating null, of s2, into s1. It then copies a null character onto the end of s1. It returns s1.

int strncmp(char *s1, char *s2, size_t n) is the same as strcmp() except it compares at most n characters.

char *strncpy(char *s1, char *s2, size_t n) is the same as strcpy() except it copies at most n characters.

char *strpbrk(char *s1, char *s2) does the same search as strcspn() except that it returns the pointer to the matching element in s1 if the element is not the terminating null. Otherwise, it returns a null pointer.

char *strrchr(char *s, int c) searches for the last occurrence of c in s and returns a pointer to it. It returns a null pointer if no match is found.

size_t strspn(char *s1, char *s2) searches for the first element in s1 that does not match any of the elements in s2. The terminating null of s2 is considered part of s2. It returns the index where the condition is true.

char *strstr(char *s1, char *s2) finds the substring of s1 that matches s2. It returns the address of the substring if found and a null pointer otherwise.