

GCC summary, AVR LibC and MSOE support

Version 2005-10-04-B

Milwaukee School of Engineering

(c) 2005

Dr.-Ing. Joerg Mossbrucker

This document contains the following:

- GCC summary
- AVR LibC and MSOE support functions

1 GCC summary

1.1 Data

1.1.1 Elementary data types

TABLE 1.1:DATA TYPES

Data type	Bits	Range
unsigned char, uint8_t	8	0... 255
char, signed char, int8_t	8	-128 ... 127
unsigned int, uint16_t	16	0 ... 65535
int, signed int, int16_t	16	-32768 ... 32767
unsigned long, uint32_t	32	0 ... 4294967295
long, signed long, int32_t	32	-2147483648 ... 2147483647
unsigned long long, uint64_t	64	0 ... 18E18
long long, signed long long, int64_t	64	-9E18 ... 9E18
float, double	32	+/-1.17E-38...+/-3.4E38

1.1.2 Constants

```
const char a=64;           // defines a char constant
#define b 55               // defines a constant, this is a
                          // pre-processor directive and does not
                          // use any memory
```

1.1.3 Scope

```
char a;                   // a is a global variable

char function(char b)    // b is a parameter for function
{                          // which will return a char
    char c;               // c is local only to function
    a=5;                  // legal since a is global
    d=6;                  // illegal, since d is local to main
}

void main()
{
    char d;               // d is local to main
    a=7;                  // legal since a is global
    c=8;                  // illegal since c is local to function
}
```

1.1.4 Storage class

```
auto int a=5;           // a is automatic

int b=6;                // b is automatic too, since this is the
                       // default storage class

static char c=6;       // static variables are allocated in
                       // memory but are local in scope
                       // static variables keep their content
                       // when the function is exited

volatile char d;       // d is a variable which can be changed
                       // by hardware outside the program
                       // this is the only storage class which
                       // will NOT be optimized away

register char e;        // e is kept in a register which allows
                       // faster access
                       // Use this at a minimum since the
                       // optimizer does a better job
```

1.1.5 Type casting

```
int a=5;
char b;
b=a; // this is not good since both variables
// are of different types
b=(char)a; // this is legal

unsigned int d; // d is 16 bit
unsigned int e=200; // e is 16 bit, hex value is C8
unsigned char f=100; // f is 8 bit, hex value is 64

d=(f*10)+e; // this will not compute correctly since
// f is char and therefore f*10 will exceed
// the range of a char, the result will be
// truncated to hex E8, which is 232
// Therefore d=232+200=432

d=((int)f*10)+e; // this will compute correctly

unsigned int g=1000;
unsigned char h;
h=(unsigned char)g; // this typecasts a bigger variable into
// a smaller one.
// Note: the result is very often
// unpredictable, in this example it is:
// g=1000 which is 3E8
// therefore h=E8 hex which is 232
// this becomes worse once the variables
// are of a signed type, since not even
// the sign might be preserved
```

1.1.6 Arrays

```
char a[10];           // a is an array, consisting of 10 elements
                    // ranging from a[0]...a[9]
char b[5]={3,6,2,1,99}; // how to initialize an array, b[0]=3
                    // b[1]=6, b[2]=2, b[3]=1, b[4]=99
char c[4][5];        // c is a two-dimensional array
char d[4][5][6];     // d is a three-dimensional array

char day[7][10]={
    "Sunday"
    "Monday"
    "Tuesday"
    "Wednesday"
    "Thursday"
    "Friday"
    "Saturday" };   // longest day name is 9 characters so
                    // array is defined with 10 characters
                    // to contain the '\0' character
```

1.1.7 Structs

```
struct date          // date is the tag name for the struct
{
    char day;
    char month;
    char year;
};

struct date birthday[10]; // declares ten birthdays of type date
birthday[5].day=15;
```

1.1.8 Unions

```
union ex {           // ex is the tag name for the union
    char character;  // all members will share a common memory
    int integer;    // allocated to the largest member of the
    long long_type; // union
};

union ex new         // declares a variable new of type ex
new.long_type=0x12345678;
                    // new.integer=0x1234
                    // new.char=0x12
```

1.1.9 Typedefs

```
typedef unsigned char byte; // declares a new variable type byte
                             // which is an unsigned char
byte a;                       // a is of type byte

typedef struct                // declares a new structure type
{
    unsigned char x;         // does not allocate memory
    unsigned char y;         // members of the struct
} point;                      // name of struct type is point

point first_point;           // first_point is of type point
first_point.x=100;           // accesses a member of first_point
```

1.1.10 Bit fields

```
struct
{
    unsigned intflag : 1;    // flag is 1 bit (0...1)
    unsigned inton : 1;      // on is 1 bit (0...1)
    unsigned intstate : 4;   // state is 4 bit (0...15)
} fsm;

fsm.flag=1;                  // only 0 and 1 are allowed as values
fsm.on=0;                    // only 0 and 1 are allowed as values
fsm.state=9;                 // values between 0 and 15 are allowed
fsm.state++;

typedef struct
{
    bit_0: 1;
    bit_1: 1;
    bit_2: 1;
    bit_3: 1;
    bit_4: 1;
    bit_5: 1;
    bit_6: 1;
    bit_7: 1;
} bits;

bits port;
port.bit_0=1;
```

1.1.11 SIZEOF operator

```
char a;
int b[10],s;
struct c
{
    char name[20];
    int id;
} d[15];

s=sizeof(int);           // return the size of int, which is 2
s=sizeof(a);            // returns the size of a, which is 1
s=sizeof(b);            // returns the size of the array b
                        // which is 10*sizeof(int)=20
s=sizeof(c)              // returns 20+2=22
s=sizeof(d)              // returns 15*(20+2)=330
```

1.1.12 Pointers

```
char *p;                 // p is a pointer to a char
char a;                  // a is a char
p=&a;                    // p is pointing to a
*p='c';                 // the memory location p is pointing to
                        // now contains 'c'

char string[10];        // string is a character array
char *p;                // p is a pointer to a char
p=string;               // p points to string[0]
p=&string[0];           // p points to string[0]

int func(char)          // test function
{
    return 10;          // returns 10
}
int b;
int (*fp)(char)         // fp is a pointer to a function having a
                        // a char as a parameter and returning
                        // an int
fp=func;                // fp is now pointing to func()
b>(*fp)(5)              // execute the function fp is pointing to
                        // with the parameter 5 and store the
                        // return value in b
```

1.2 Expressions

1.2.1 Operators

TABLE 1.2: ARITHMETIC OPERATORS

Operation	symbol
Multiply	*
Divide	/
Addition	+
Subtraction (binary minus)	-
Negation (unary minus)	-
Modulo	%

TABLE 1.3: BITWISE OPERATORS

Operation	Symbol
Ones complement	~
Bitwise AND	&
Bitwise OR	
Bitwise EXOR	^
Left shift	<<
Right shift	>>

TABLE 1.4: LOGICAL OPERATORS

Operation	Symbol
AND	&&
OR	

TABLE 1.5: RELATIONAL OPERATORS

Operation	Symbol
Is equal to	==
Is not equal to	!=
Greater than	>
Greater than or equal	>=
Less than	<
Less than or equal	<=

TABLE 1.6: CREMENTAL OPERATORS

Cremental operators	
Post-increment	x++
Post-decrement	x--
Pre-increment	++x
Pre-decrement	--x

1.2.2 Operator precedence

TABLE 1.7: OPERATOR PRECEDENCE

Name	Level	Operators	Grouping
Primary	1 (high)	() . [] ->	Left to right
Unary	2	! ~ - (type) * & ++ -- sizeof	Right to left
Binary	3	* / %	Left to right
Arithmetic	4	+ -	Left to right
Shift	5	<< >>	Left to right
Relational	6	< <= > >=	Left to right
Equality	7	== !=	Left to right
Bitwise	8	&	Left to right
Bitwise	9	^	Left to right
Bitwise	10		Left to right
Logical	11	&&	Left to right
Logical	12		Left to right
Conditional	13	?:	Right to left
Assignment	14 (low)	= += -= /= *= %= <<= >>= &= ^= =	Right to left

1.3.4 IF

```
if(expression)
{
    statement 1;          // if expression is true statements 1 and 2
    statement 2;          // are executed
}

if(expression)           // if expression is true statement
    statement;           // is executed
```

1.3.5 IF ELSE

```
if(expression)
{
    statement 1;          // if expression is true statements 1 and 2
    statement 2;          // are executed
}
else
{
    statement 3;          // otherwise statements 3 and 4
    statement 4;          // are executed
}

if(expression)
    statement 1;          // if expression is true statement 1
else
    statement 2;          // is executed
                        // otherwise statement 2 is executed
```

1.3.6 SWITCH CASE

```
switch(expression)      // expression is evaluated
{
    case const 1:        // first result
        statement 1;
        statement 2;
        break;          // without break statement 3 ff. will also
    case const 2:        // be executed
        statement 3;
        statement 4;
        break;          // without break statement 5 ff. will also
    default:             // be executed
        statement 5;
        statement 6;
}
```

1.3.7 BREAK

```
while(1)                // forever loop
{
    while(1)            // forever loop
    {
        if(c>10)
            break;      // this breaks the inner while loop
        c++;           // and continues the outer loop again
    }
}
```

1.3.8 CONTINUE

```
while(1)                // forever loop
{
    while(1)            // forever loop
    {
        if(c>10)
            continue;  // will skip the rest of the inner while
        c++;           // loop and continues with the inner while
    }
}
```

1.4 Functions

```
void func1(void);       // function prototype
int func2(char);       // function prototype

main()                  // main function
{
    statement 1;
    statement 2;
}

void func1(void)        // function definition
{                       // must match prototype
    statement 3;
    statement 4;
}

int func2(char x)      // function definition
{                       // must match prototype
    statement 5;
    statement 6;
}
```

1.5 Pre-processor directives

1.5.1 #include

```
#include <filename>    // will include filename by looking first
                       // in the standard library set
#include "filename"    // will include filename by looking first
                       // in the same directory as source
```

1.5.2 #define, undefine

```
#define ten 10
                       // will replace every occurrence of ten with 10

#undefine ten
                       // will remove the defined term ten
```

1.5.3 #ifdef, #ifndef, #else, #endif

```
#ifdef name
  set of statements 1 // will be executed if name is defined
#else
  set of statements 2 // will be executed if name is not defined
#endif

#ifndef name
  set of statements // will be executed if name is not defined
#endif

#if expression1
  set of statements 1 // executed if expression1 is true
#elif expression2
  set of statements 2 // else executed if expression2 is true
#else
  set of statements 3
#endif
```

2 AVR LibC and MSOE functions

2.0.1 Register and Port IO in C

```
#include <avr/io.h>    // this include file is needed

PORTB = 10;           // every register can be accessed with its
TCNT0 = 50;           // name

#include <MSOE/bit.c>  // this include file is needed

sbi(PORTA,4);         // set bit 4 of PORTA
cbi(PORTB,3);         // clear bit 3 of PORTB

if(tbi(PINB,5)==1)    // tests if bit 5 of PINB is set

if(tbi(PINB,1)==0)    // tests if bit 1 of PINB is clear
```

2.0.2 Register and Port IO in assembly

```
#define _SFR_ASM_COMPAT 1    // this is needed for every assembly
                             // source file so we can use the IO
                             // names as usual

#define __SFR_OFFSET 0      // this is needed for every assembly
                             // source file so we can use IO with
                             // IN and OUT statements

#include <avr/io.h>         // this will automatically include
                             // the right file for the device
                             // since this file is processed by the
                             // C-preprocessor comments after #include
                             // or #define must be preceded by //

                             ; afterwards we can use the traditional
                             ; semicolon

out  PORTB,r20             ; output r20 on PORTB
in   r20,PORTB            ; input PORTB to r20
cbi  PORTB,6              ; clear bit 6 of PORTB
sbi  DDRA,DDA7            ; set DDA7 bit in DDRA
```

2.0.3 Interrupts and Signals in C

```
#include <avr/signal.h>      // required include file
#include <avr/interrupt.h>   // required include file

INTERRUPT(SIG_ADC)         // this ISR can be interrupted
{
    // Interrupt service routine
}

or

SIGNAL(SIG_ADC)           // this ISR cannot be interrupted
{
    // Interrupt service routine
}
```

2.0.4 Interrupt control in C

```
sei();           // enables interrupts globally
cli();          // disables interrupts globally
```

2.0.5 Interrupts in assembly

```
.global SIG_OVERFLOW1      ; interrupt handlers in assembly need to
                           ; be defined global
SIG_OVERFLOW1:             ; entry point for interrupt handler for
                           ; the timer1 overflow interrupt

    out  PORTB,r20         ; this is just an example
    out  PORTB,r19         ; dto.

    reti                  ; interrupt handler terminates with a reti
```

2.0.6 Interrupt control in assembly

```
sei                ; enables interrupts globally
cli                ; disables interrupts globally
```

2.0.7 Interrupt Service Routine pre-defined names

TABLE 2.1:ISR NAMES

Signal Name	Description
SIG_2WIRE_SERIAL	2-wire Serial Interface (aka I2C)
SIG_ADC	ADC Conversion Complete
SIG_COMPARATOR	Analog Comparator Interrupt
SIG_EEPROM_READY	EEPROM Ready Interrupt
SIG_INPUT_CAPTURE1	Input Capture1 Interrupt
SIG_INPUT_CAPTURE3	Input Capture3 Interrupt
SIG_INTERRUPT0	External Interrupt0
SIG_INTERRUPT1	External Interrupt1
SIG_INTERRUPT2	External Interrupt2
SIG_INTERRUPT3	External Interrupt3
SIG_INTERRUPT4	External Interrupt4
SIG_INTERRUPT5	External Interrupt5
SIG_INTERRUPT6	External Interrupt6
SIG_INTERRUPT7	External Interrupt7
SIG_OUTPUT_COMPARE0	Output Compare0 Interrupt
SIG_OUTPUT_COMPARE1A	Output Compare1(A) Interrupt
SIG_OUTPUT_COMPARE1B	Output Compare1(B) Interrupt
SIG_OUTPUT_COMPARE1C	Output Compare1(C) Interrupt
SIG_OUTPUT_COMPARE2	Output Compare2 Interrupt
SIG_OUTPUT_COMPARE3A	Output Compare3(A) Interrupt
SIG_OUTPUT_COMPARE3B	Output Compare3(B) Interrupt
SIG_OUTPUT_COMPARE3C	Output Compare3(C) Interrupt
SIG_OVERFLOW0	Overflow0 Interrupt
SIG_OVERFLOW1	Overflow1 Interrupt
SIG_OVERFLOW2	Overflow2 Interrupt
SIG_OVERFLOW3	Overflow3 Interrupt
SIG_PIN	
SIG_PIN_CHANGE0	
SIG_PIN_CHANGE1	
SIG_RDMAC	
SIG_SPI	SPI Interrupt
SIG_SPM_READY	Store Program Memory Ready

TABLE 2.1:ISR NAMES

Signal Name	Description
SIG_SUSPEND_RESUME	
SIG_TDMAC	
SIG_UART0	
SIG_UART0_DATA	UART(0) Data Register Empty Interrupt
SIG_UART0_RECV	UART(0) Receive Complete Interrupt
SIG_UART0_TRANS	UART(0) Transmit Complete Interrupt
SIG_UART1	
SIG_UART1_DATA	UART(1) Data Register Empty Interrupt
SIG_UART1_RECV	UART(1) Receive Complete Interrupt
SIG_UART1_TRANS	UART(1) Transmit Complete Interrupt
SIG_UART_DATA	UART Data Register Empty Interrupt
SIG_UART_RECV	UART Receive Complete Interrupt
SIG_UART_TRANS	UART Transmit Complete Interrupt
SIG_USART0_DATA	USART(0) Data Register Empty Interrupt
SIG_USART0_RECV	USART(0) Receive Complete Interrupt
SIG_USART0_TRANS	USART(0) Transmit Complete Interrupt
SIG_USART1_DATA	USART(1) Data Register Empty Interrupt
SIG_USART1_RECV	USART(1) Receive Complete Interrupt
SIG_USART1_TRANS	USART(1) Transmit Complete Interrupt
SIG_USB_HW	

Please note that the pre-processor and compiler do NOT spell-check names given in ISR (neither in C nor in assembly).

2.0.8 Delay functions

- required includes:

```
#include <MSOE/delay.c>           // this include file is required
```

- in delay.c:

```
void delay_ms(uint16_t ms);       // delays ms milliseconds
void delay_us(uint16_t us);       // delays us microseconds
```

- in globaldef.h

```
#define F_CPU          14745600L   // 14.7456MHz processor
```

2.0.9 LCD control

- required includes:

```
#include <MSOE/lcd.c>           // this include file is required
#include <MSOE/delay.c>        // this include file is required
```

- in lcd.c:

```
// high level functions
void lcd_init(void);           // initialize LCD
void lcd_home(void);          // set cursor to home
void lcd_clear(void);         // clear display
void lcd_goto_xy(uint8_t x,uint8_t y); // set cursor position
void lcd_print_string(char *string); // print string at current
                                     // position
void lcd_print_hex(uint8_t hex); // print hex number on LCD
void lcd_print_uint8(uint8_t no); // print uint8 on LCD
void lcd_print_int8(int8_t no);  // print int8 on LCD
void lcd_print_uint16(uint16_t no); // print uint16 on LCD
void lcd_print_int16(int16_t no); // print int16 on LCD
void lcd_print_float(float no);  // print float on LCD
```

```
// implements a simple printf for the LCD
// supported formats:
// %d,%i signed 16bit integer
// %c character
// %f float
// %s null-terminated string
// %o octal number
// %x hex number
// %u unsigned 16bit integer
// %% %
//
// doubles are not supported (floats and doubles are the
// same for gcc for the AVR)
// no formatting is implemented
// \n \t etc not yet supported
//
void lcd_printf(char *fmt, ...)
```

- in lcdconf.h:

```
#define LCD_CTRL_PORT PORTB
#define LCD_CTRL_DDR DDRB
#define LCD_CTRL_RS 2
#define LCD_CTRL_RW 1
#define LCD_CTRL_E 0
```

```
#define LCD_DATA_PORT    PORTB
#define LCD_DATA_DDR    DDRB
```

2.0.10 UART control

- required includes:

```
#include <MSOE/uart.c>           // this include file is needed
```

- in uart.c:

```
void uart_init(void)             // initilaizes UART
void uart_send_byte(uint8_t tx_char) // sends byte via UART
uint8_t uart_receive_byte(void)  // returns byte received by UART
```

- in uart.h:

```
#define UART_BAUD_RATE    9600   // this defines the baudrate
```

- in globaldef.h

```
#define F_CPU             14745600L // 14.7456MHz processor
```

2.0.11 Keypad control

| This section under development.

2.0.12 External PC keyboard control

| This section under development.

2.0.13 Bootloader support

See section 5.1 of avr-libc-user-manual.

2.0.14 EEPROM support

See section 5.2 of avr-libc-user-manual.

2.0.15 AVR device specific IO definitions

```
#include <avr/sfr_defs.h>    // this include file is needed

/* The following macros then are defined

RAMEND - constant describing the last on-chip RAM location

XRAMEND - constant describing the last possible RAM location

E2END - constant describing the address of the last EEPROM

FLASHEND- constant describing last byte address in FLASH

SPM_PAGESIZE- flash pagesize for devices with bootloader support

*/
```

2.0.16 Program space string utilities

See section 5.4 of avr-libc-user-manual.

2.0.17 Power management

See section 5.6 of avr-libc-user-manual.

2.0.18 Watchdog timer

See section 5.7 of avr-libc-user-manual.

2.0.19 Character operations

See section 5.8 of avr-libc-user-manual.

2.0.20 System errors

See section 5.9 of avr-libc-user-manual.

2.0.21 Mathematics

See section 5.11 of avr-libc-user-manual.

2.0.22 General utilities

See section 5.14 of avr-libc-user-manual.

2.0.23 Strings

See section 5.15 of avr-libc-user-manual.

2.0.24 Interrupts and Signals

See section 5.16 of avr-libc-user-manual.

2.0.25 Special function registers

See section 5.17 of avr-libc-user-manual.

2.0.26 Inline Assembler

See section 7.4 of avr-libc-user-manual.

2.0.27 Passing parameters between C and ASM

See section 7.3.14 of avr-libc-user-manual.