USB MASTER

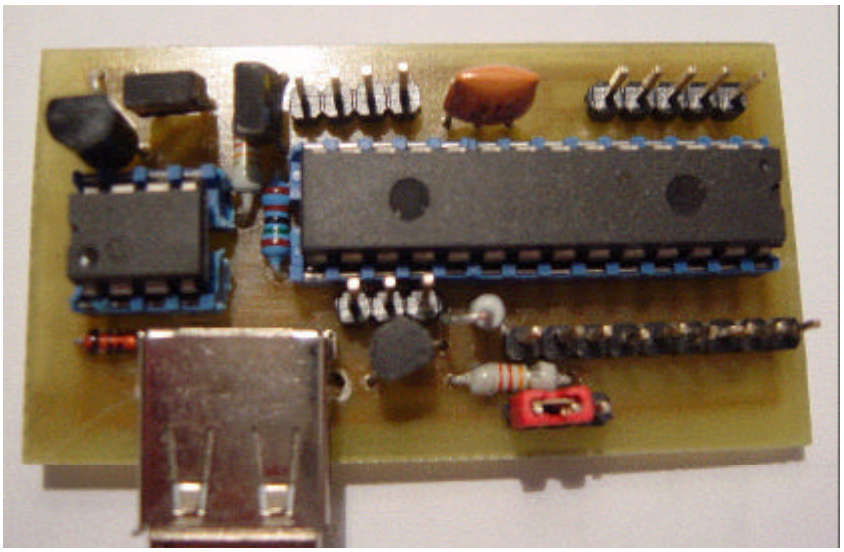
USB MacroMaster and USB BigMaster

1 Overview

The USB Master is a low cost, single board computer based on the PIC 16C745 or PIC 16C765 microcontroller. USB Masters are applicable for pilot projects and for applications with low production volumes. A USB Master is programmed using assembler code. The size of a USB Master is about 25 mm x 50 mm.

Presently there are two USB Master versions, one is called **USB MacroMaster**, it has 16 input/ output pins and is based on a PIC 16C745 microcontroller. The other one is called **USB BigMaster**, it has 27 input/ output pins and is based on a PIC 16C765 microcontroller. A USB Master is controlled, programmed und debugged via a USB interface. **USB MacroMaster** and **USB BigMaster** are further extensions of the existing Master product line with **µ-Master** and **MacroMaster.** µ-Master and MacroMaster have 8 and 16 I/ O pins, respectively, and are controlled, programmed und debugged via a serial interface.

A development software works with all current Windows® versions and is freely available as download on www.spengergasse.at/~tarkany. Windows98® requires to install a USB driver. The minimum hardware requirements are a Pentium II and USB support on the motherboard. The development software supports both USB Master versions.

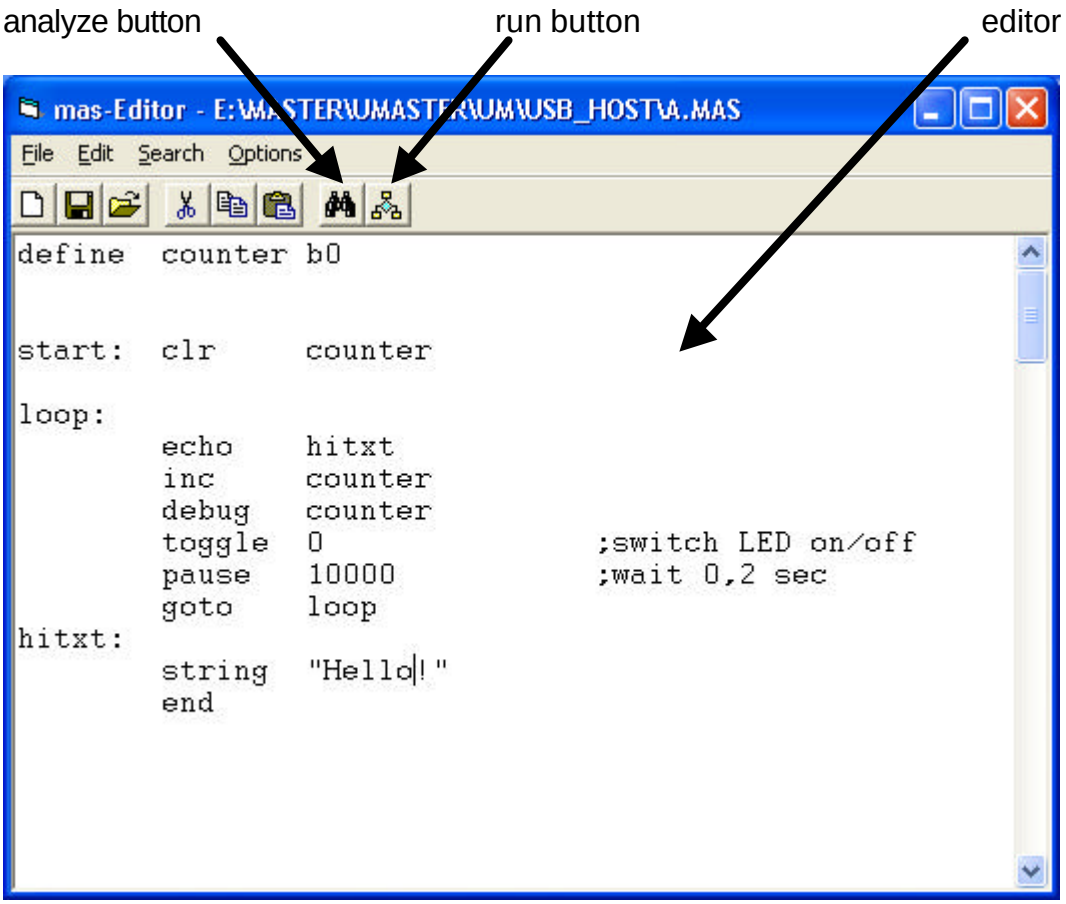


2 USB Master Development System

The USB Master development system allows the developer to observe all hardware and software conditions of the USB Master. The development software includes an ASCII editor, an assembler, a downloader and a debugger.

2.1 Editor

The following figure shows the first window of the development system, a basic editor. Functions like new, save, open, cat, paste and copy can be started from the toolbar.
The **analyze button** starts the analyzer window, that allows to read and write the RAM, read and write the EEROM and disassemble the code stored in a USB Master.
The **run button** starts the assembler and downloads a new program code from a PC to a USB Master.

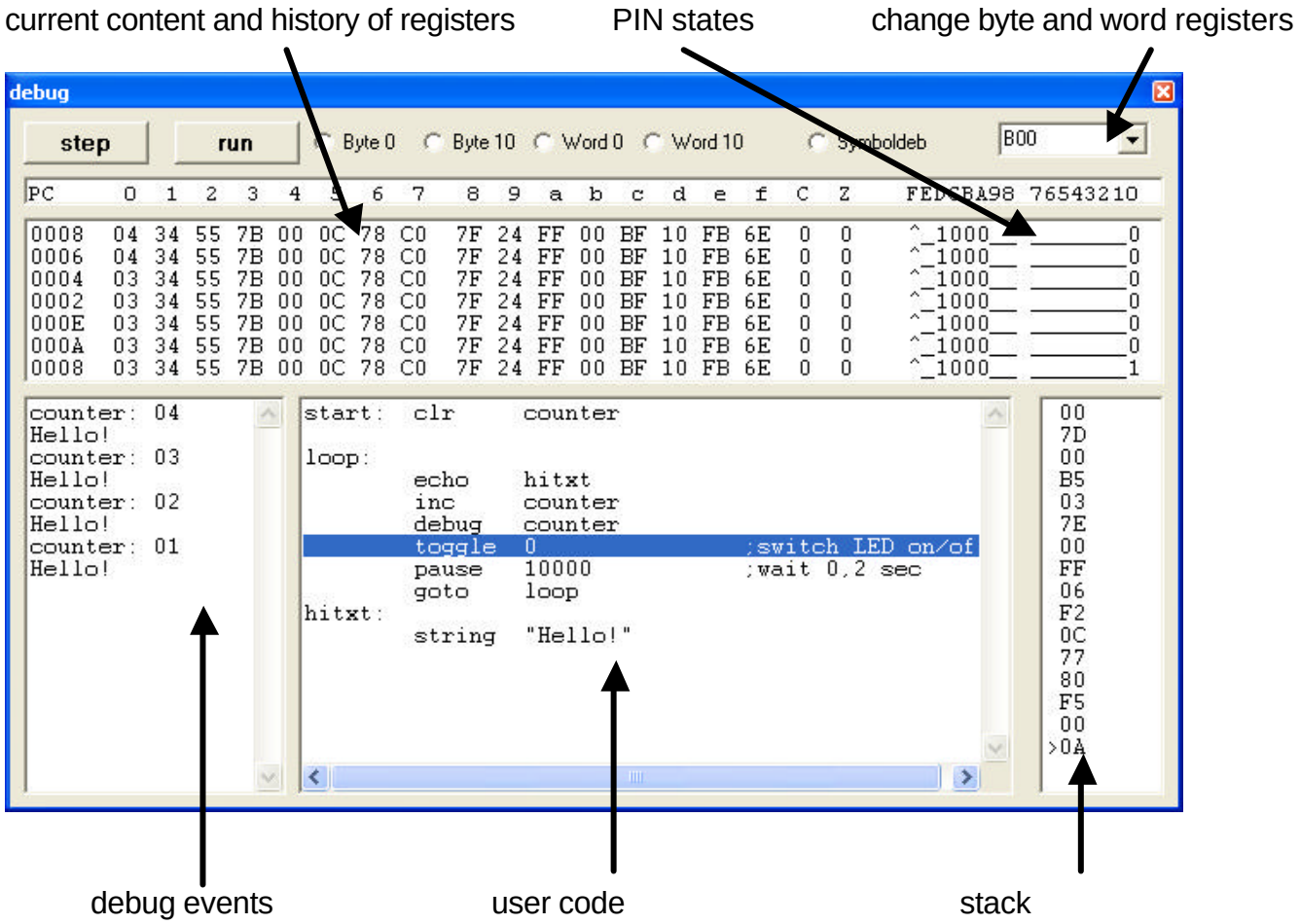


2.2 Debugger

A developer can trace his own program in the debug window. He can start and stop his software application, observe the current content of the registers and the history of the registers.
The developer can change the content of any register and use symbolic debugging.

The four possible states of the I/O pins are displayed as follows:

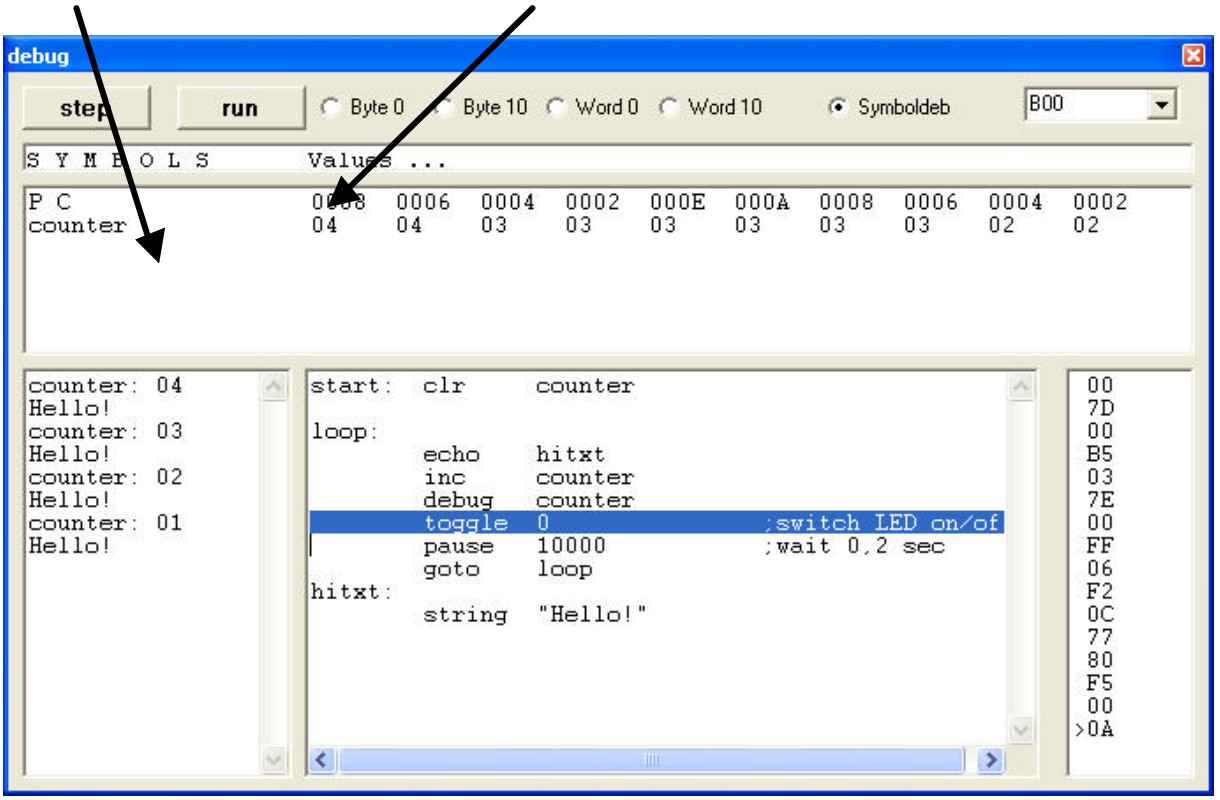
Input	- high	^
Input	- low	_
Output	- high	1
Output	- low	0



The content of the registers can be displayed as bytes (Byte 0: byte 0 to 15, Byte 10: byte 16 to 31) and words (Word 0: word 0 to 15, Word 10: word 16 to 31).
Remark: byte 0 to 31 corresponds to word 0 to 15.

Symbolic Debugger

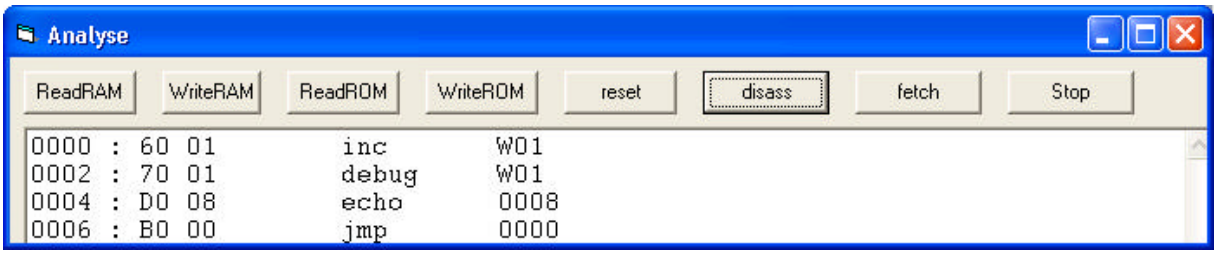
user defined symbols current value and history of symbols



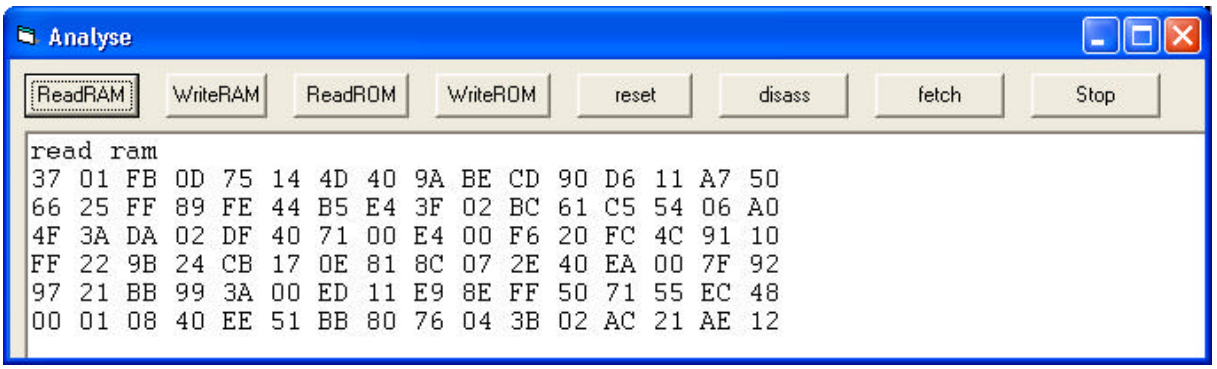
Developers can define symbols in their software code and then trace the symbols in the debug window.

2.3 Analyzer

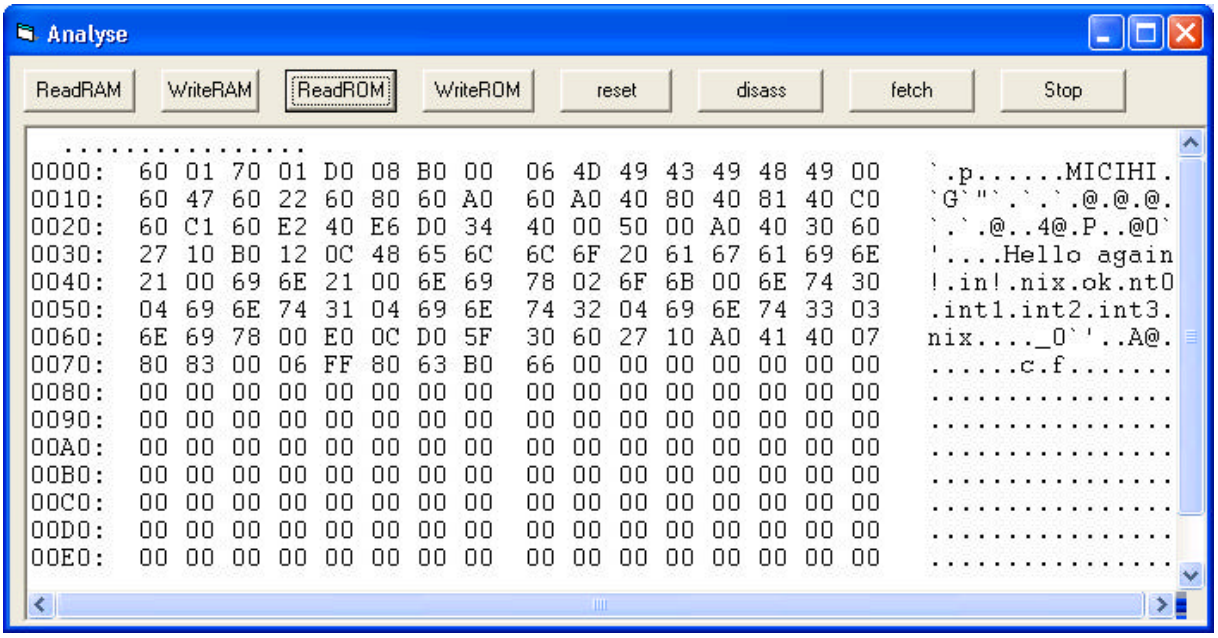
The **analyze button** starts the analyzer window.
A developer can read and write the RAM, read and write the EEROM and disassemble the program code currently stored in a USB Master.



Disassembler



Read RAM Content



Read ROM Content

3 Physical Characteristics

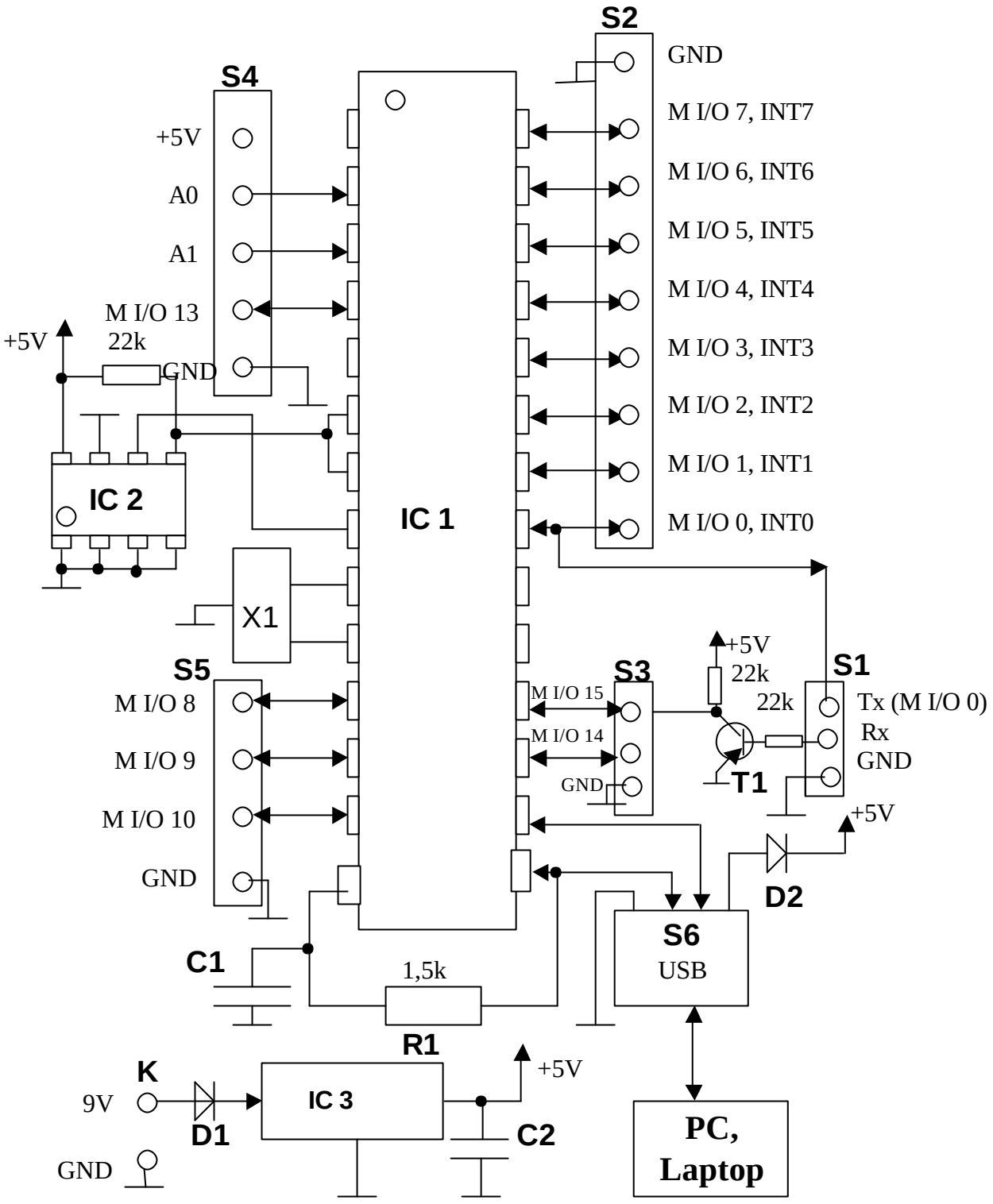
A USB Master consists of two main components, a PIC 16C745 or PIC 16C765 microcontroller which executes the software and a LS 2416 program memory. The LS 2416 EEPROM, a small microchip with 8 pins, stores the application program that is fetched and executed by the microcontroller. Other components are a 6 MHz resonator, a 5 V power supply and connectors for 16 (USB MacroMaster) or 27 (USB MacroMaster) input/ output pins.

3.1 Hardware Features

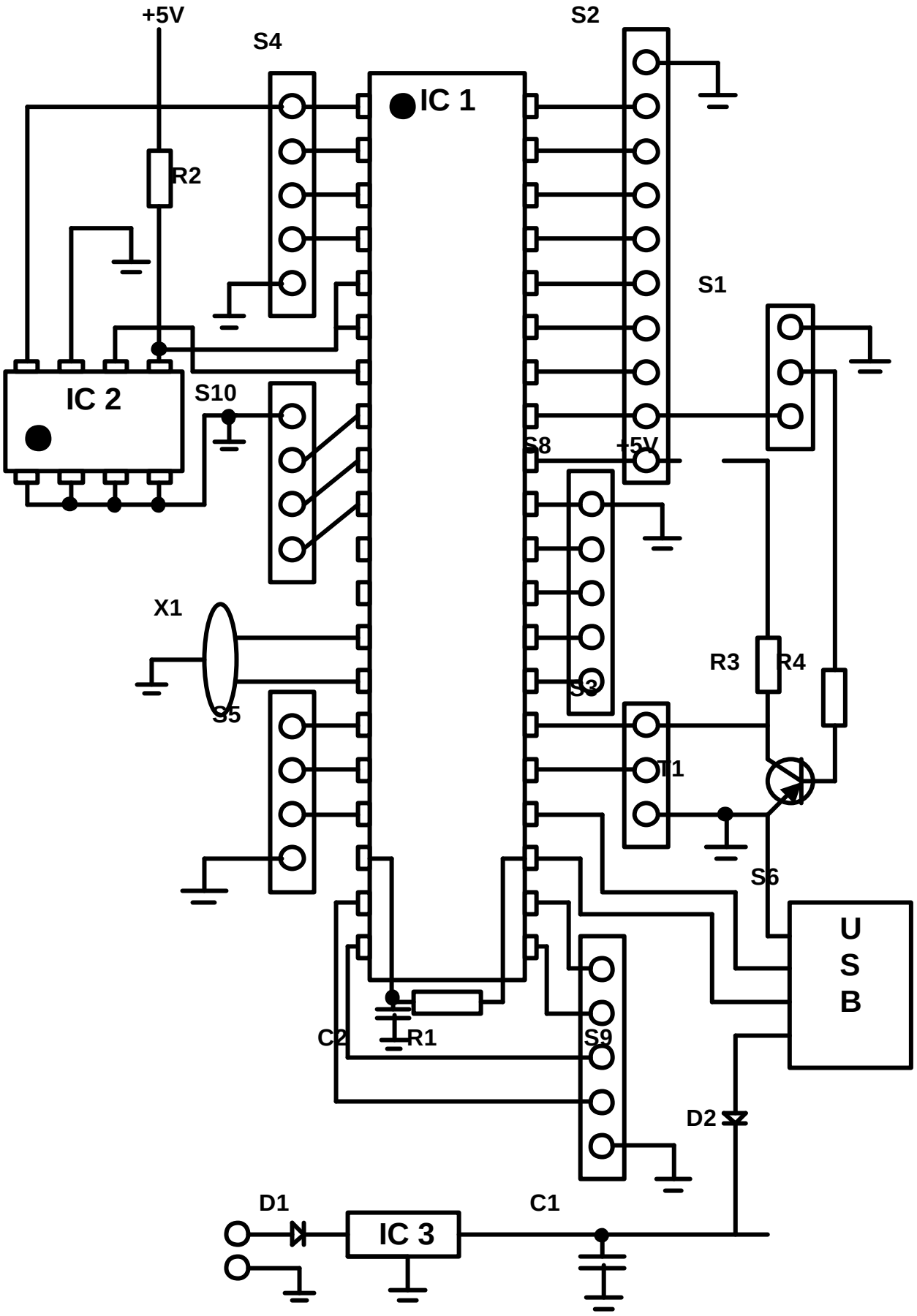
- power supply voltage 9 - 13 V
- power supply current 16 mA - 70 mA
- I/ O pin maximum sink/ source current 25/ 25 mA
- processor clock 6 MHz
- highest/ lowest input voltage 6,2/ -6,2 V

- **USB MacroMaster:** 14 programmable digital multifunctional I/Os (M I/Os)
- **USB BigMaster:** 25 programmable digital multifunctional I/Os (M I/Os)
- 2 analog inputs with A/D converter (8 bits)
- 2 analog outputs (M I/O 9 and 10) with Pulse Width Modulation (PWM) (8 bits)
- RX serial receive pin (M I/O 15) with buffered FIFO registers (8 bytes)
- TX serial transmit pin (USB Master use M I/O 0 but any digital M I/O pin could be used for serial transmission)
- RC5 transmitter(any M I/O can be used) / receiver (M I/O 8)
- 8 vector interrupts (M I/O 0 - 7)
- 2 kB EEPROM
- 6 MHz external clock
- user programmable USB interface
- CPU emulation

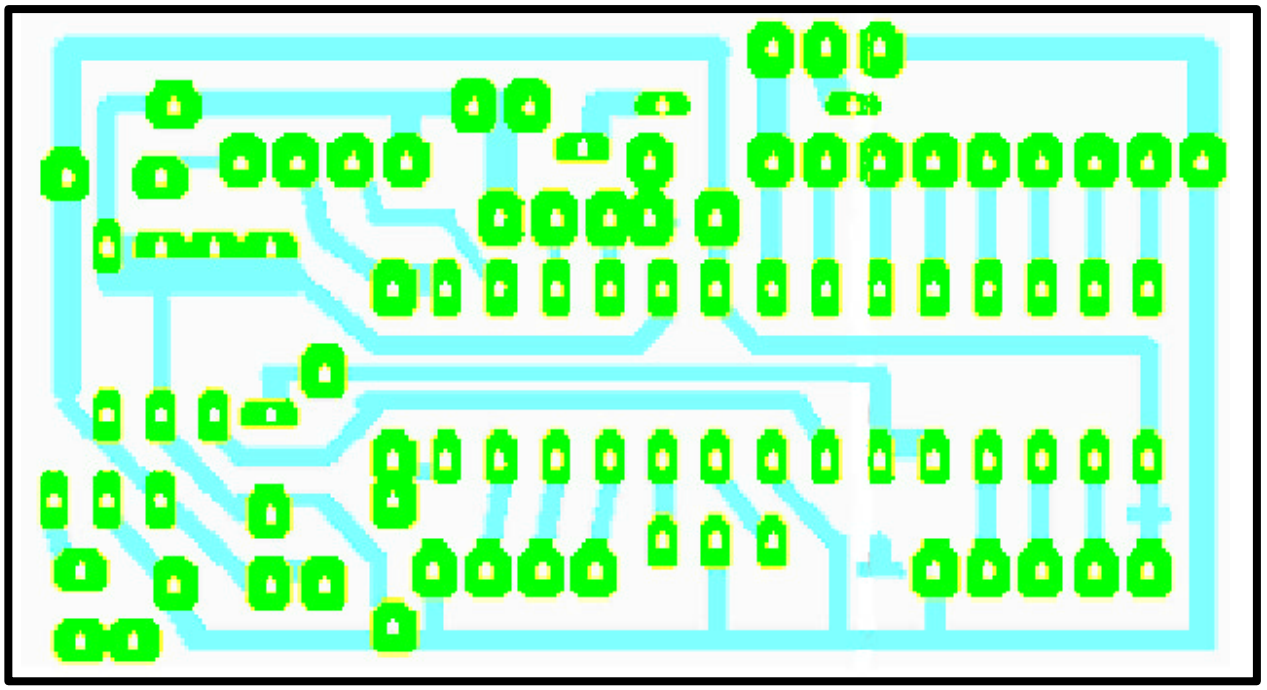
3.2 USB MacroMaster Schematic Diagram



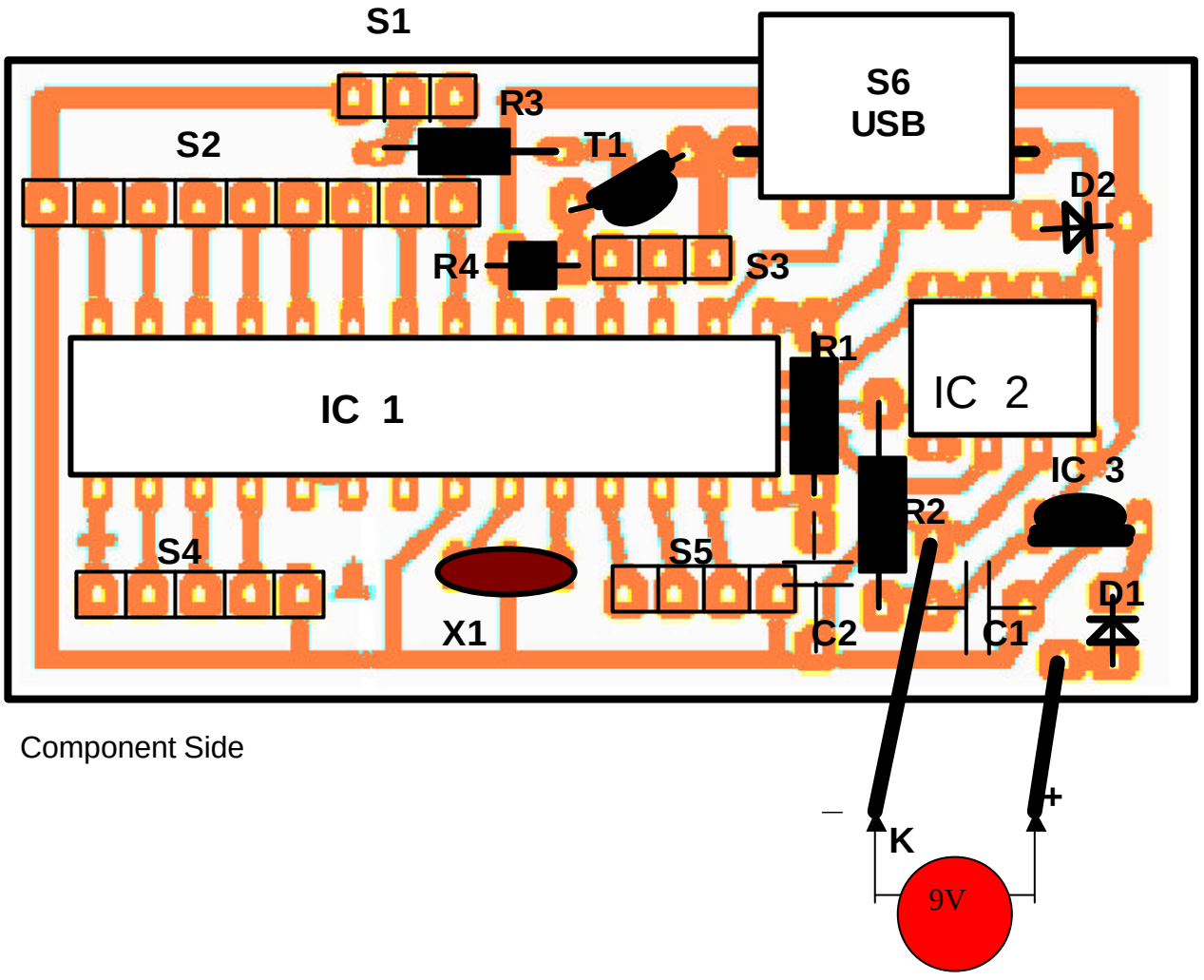
3.3 USB BigMaster Schematic Diagram



3.4 USB MacroMaster Hardware Layout

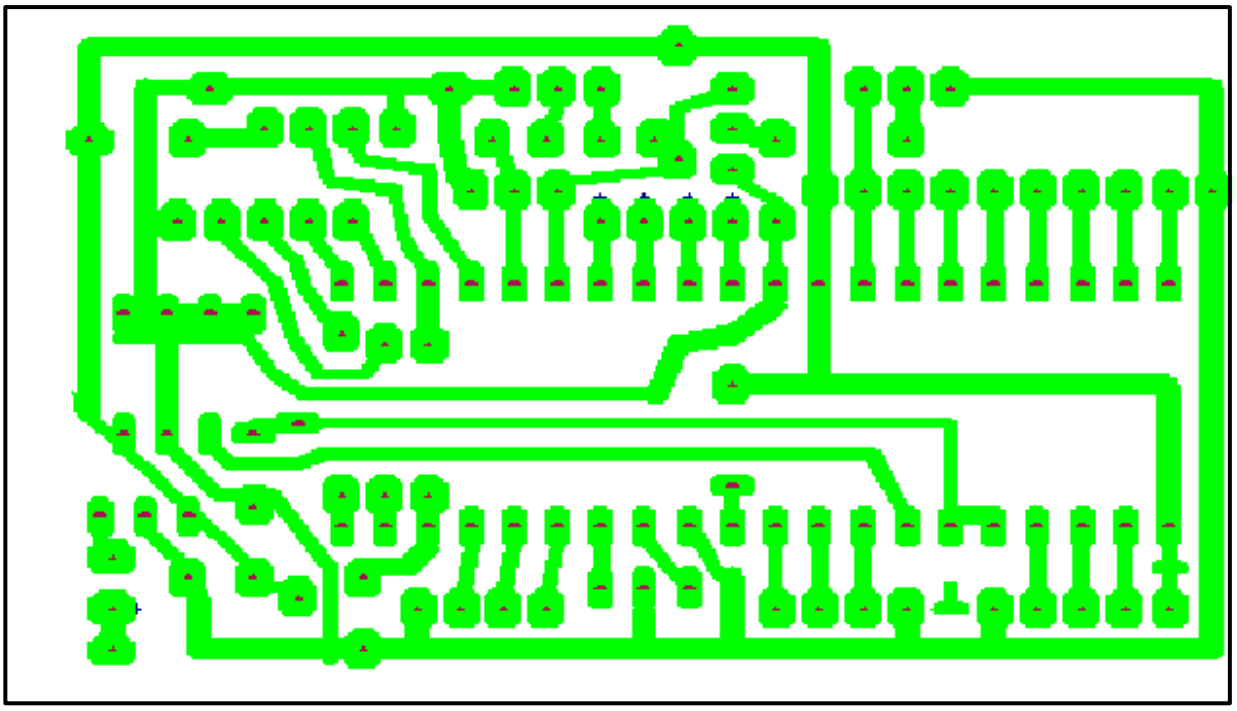


Soldering Side

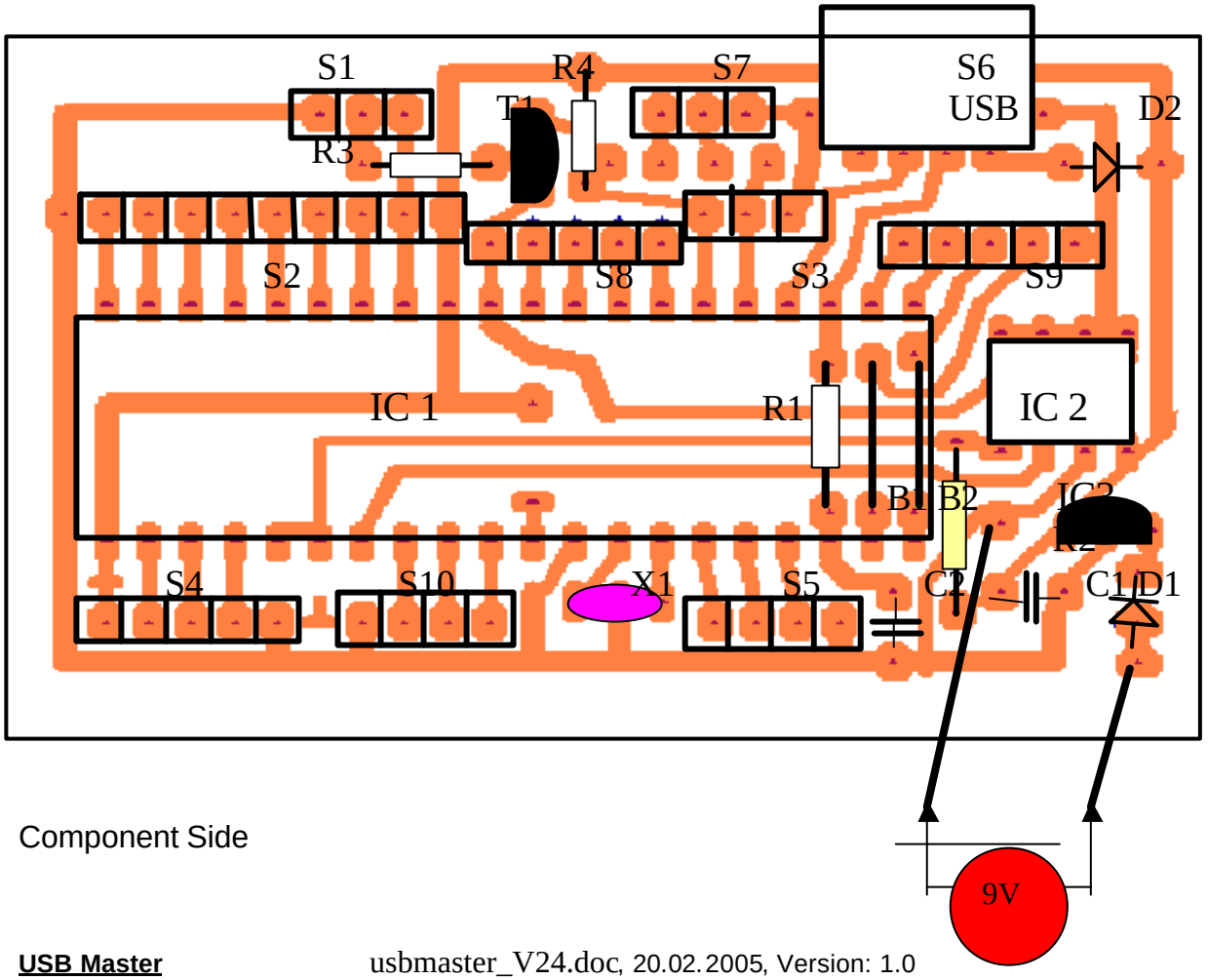


Component Side

3.5 USB BigMaster Hardware Layout



Soldering Side

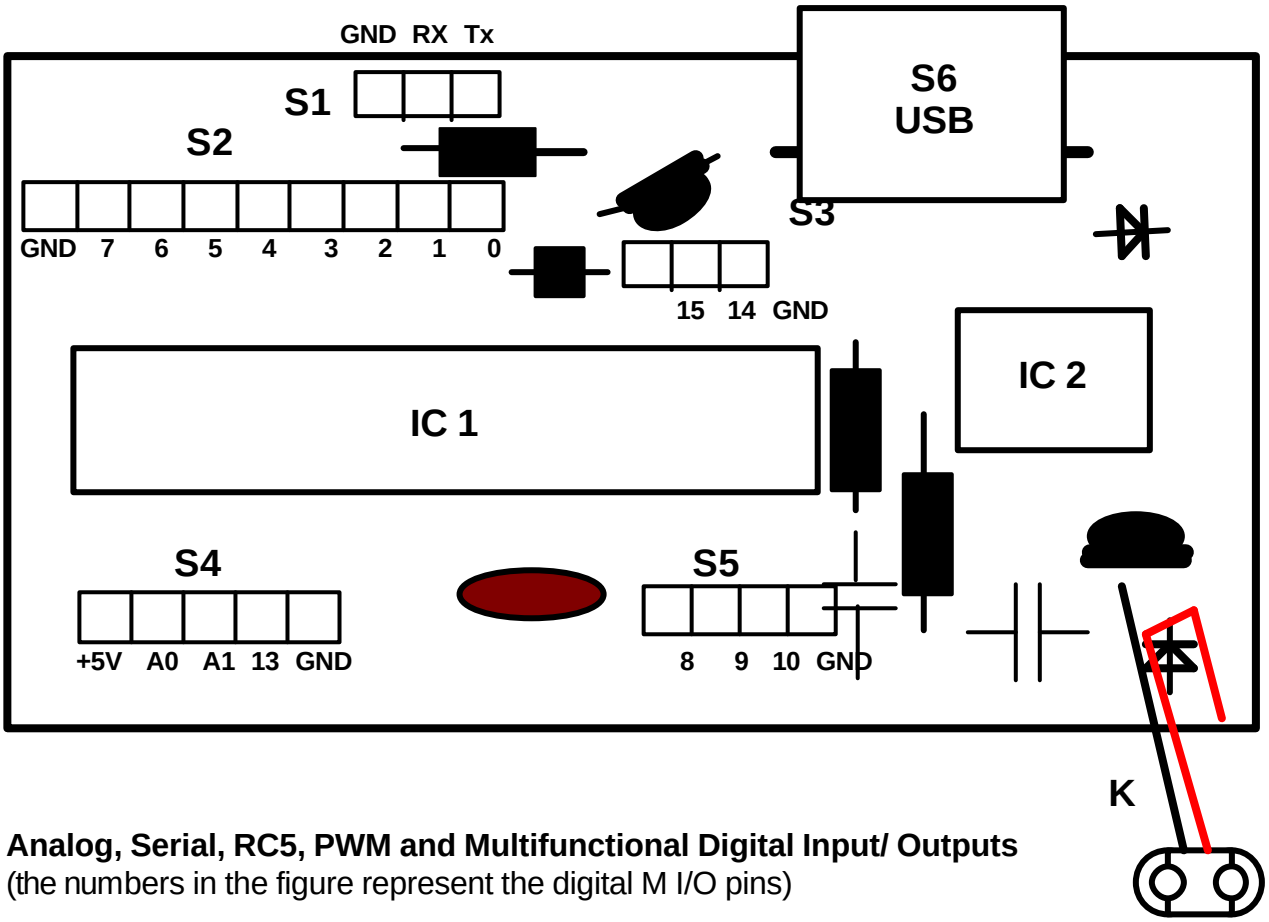


Component Side

3.6 USB MacroMaster and BigMaster Electrical Components

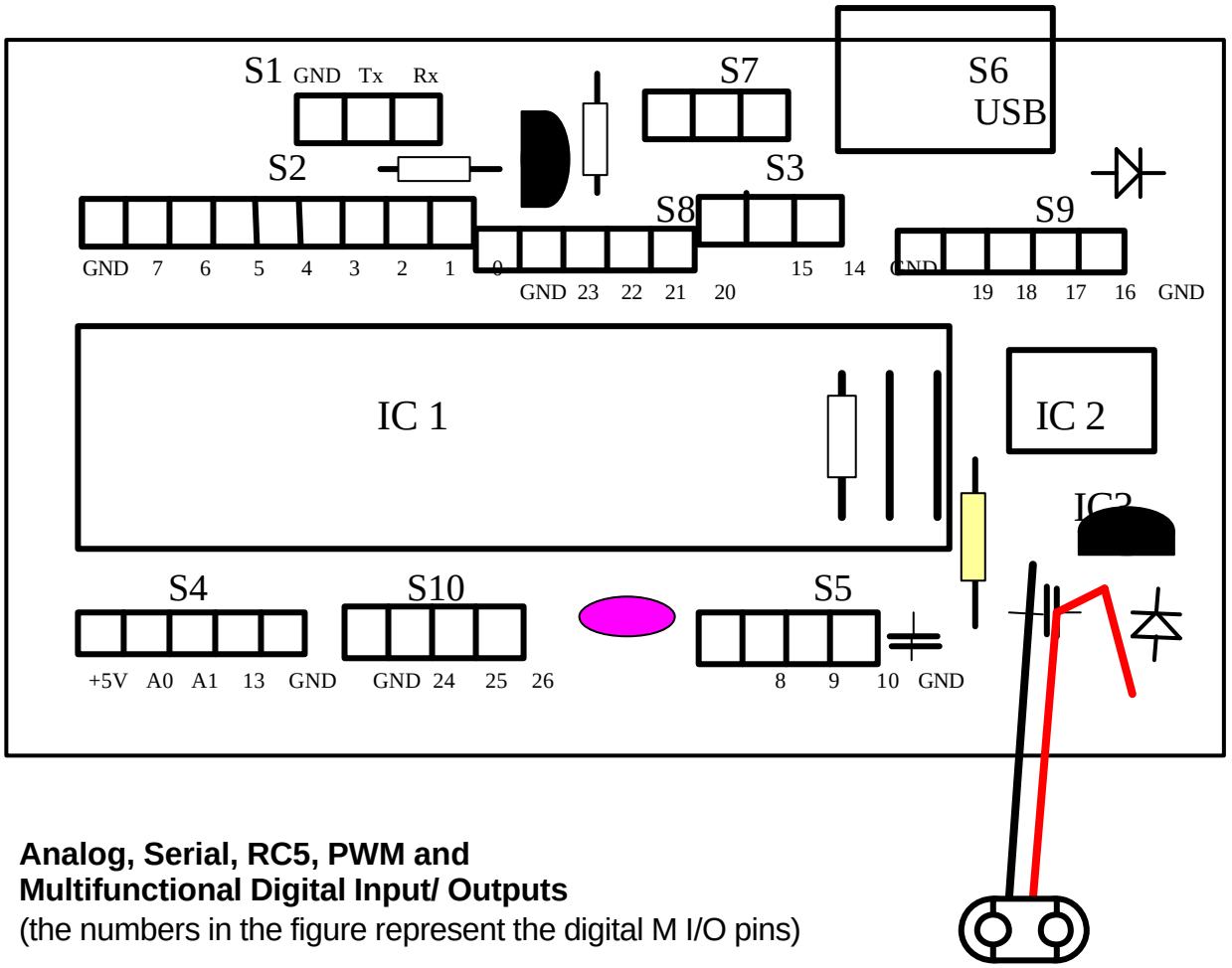
IC 1	PIC 16C745	microcontroller USB MacroMaster
IC 1	PIC 16C765	microcontroller USB BigMaster
IC 2	LS 2416	2 kByte EEPROM
IC 3	uA 78 05	5 V voltage stabilization
T1	transistor	npn
X1	resonator	6 MHz
R1	resistor	1,5 kOhm
R2, R3, R4	resistor	22 kOhm
D1, D2	diode	
C1, C2	capacitor	10 - 100 nF
S1	serial interface connector	3 pins
S2	M I/O connector	9 pins
S3	M I/O connector	3 pins
S4	analog input, M I/O connector	5 pins
S5	PWM, M I/O connector	4 pins
S6	USB connector	
S7	M I/O connector	3 pins (USB BigMaster only)
S8	M I/O connector	5 pins (USB BigMaster only)
S9	M I/O connector	5 pins (USB BigMaster only)
S10	M I/O connector	4 pins (USB BigMaster only)
B1, B2	Bridges	(USB BigMaster only)
K	9 V battery connector	2 pins

3.7 USB MacroMaster Input/ Output Layout



Analog, Serial, RC5, PWM and Multifunctional Digital Input/ Outputs
(the numbers in the figure represent the digital M I/O pins)

3.8 USB BigMaster Input/ Output Layout



Analog, Serial, RC5, PWM and
Multifunctional Digital Input/ Outputs
(the numbers in the figure represent the digital M I/O pins)

3.9 Special I/ Os

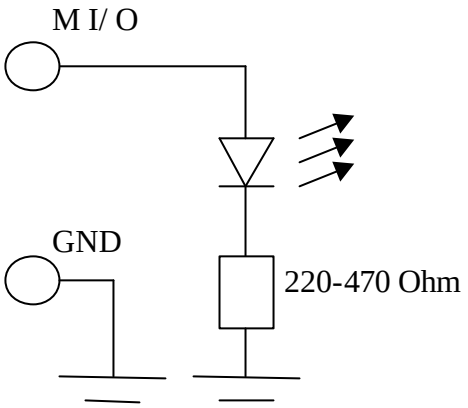
A0, A1 (S4)	-	analog inputs
RX (S3)	-	asynchronous serial input, receive pin
TX (S3)	-	asynchronous serial output, transmit pin (identical with M I/O pin 0)
RC5 input (S5)	-	identical with M I/O 8 (Ron's Code, an infrared data transmission code)
PWM 0 (S5)	-	analog output, identical with M I/O 9
PWM 1 (S5)	-	analog output, identical with M I/O 10

4 How to Use the M I/ O Pins

M I/O Pins are very sensitive, they are MOSFET inputs/ outputs and you can destroy them through improper operation.
Do not connect these pins directly to electrical motors and electrical lamps, the pins are too weak to switch these components. For these purposes it is necessary to use amplifiers.
Each USB Master M I/O pins can sink or source only 25 mA. However, the sum of all sink and source currents for all pins should not exceed 50 mA.

4.1 LED Control Circuit Diagram

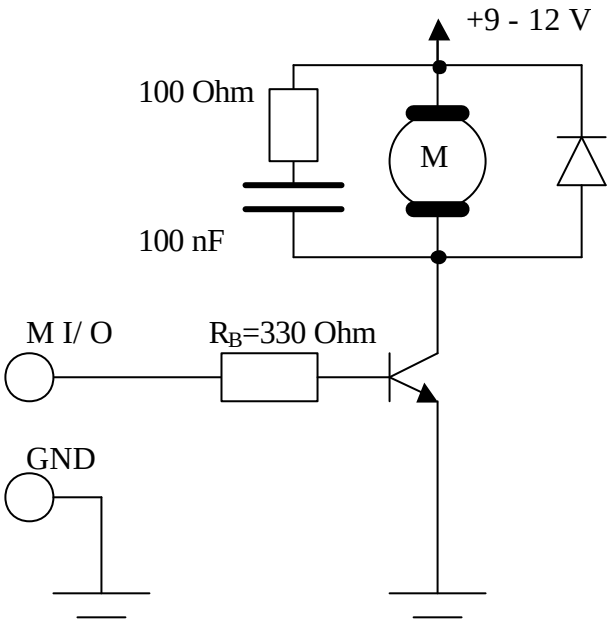
The following circuit diagram shows how to connect a LED to a USB Master.
A resistor limits the source current, without the resistor the M I/O pin would be destroyed.
The following instructions can control the state of the LED:
HIGH, LOW, TOGGLE, RCSEND.



Calculation for $U_{I/O}$ = 5 V, I_{LED} ~ 10 mA, U_{LED} ~ 2 V:
R ~ ($U_{I/O}$ – U_{LED}) / I_{LED} = (5 – 2) / 10 = 300 Ohm

4.2 Motor Control Circuit Diagram

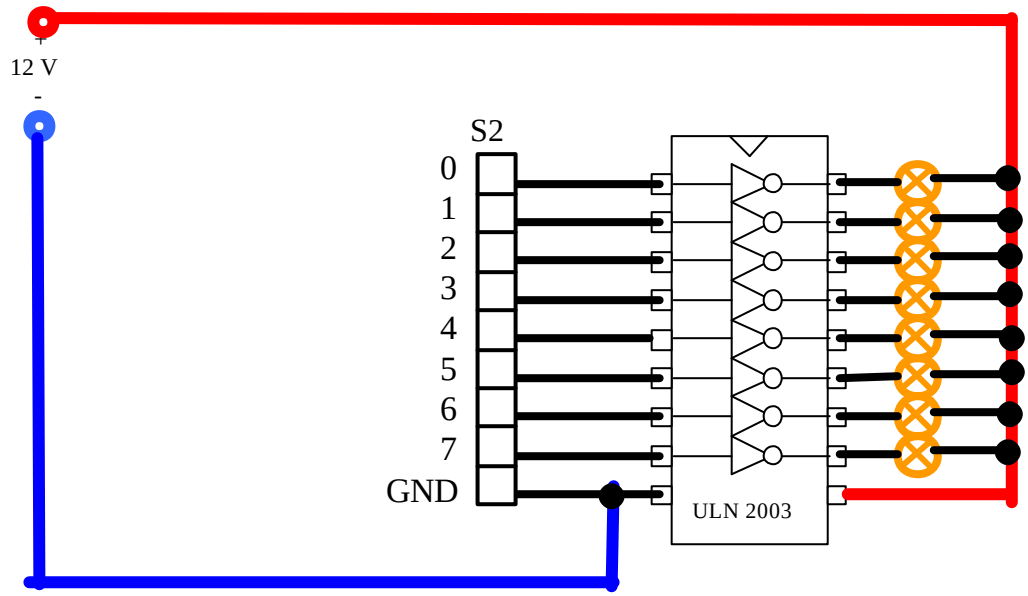
The following instructions can switch the motor:
HIGH, LOW, TOGGLE, RCSEND.



Calculation for $U_{I/O}$ = 5 V, U_M = 12 V, R_B = 330 Ohm, U_{CE} ~ 0 V, U_B ~ 0,6 V, $B_{Transitor}$ >= 750 (for example a Darlington Transistor BD 643):
$I_B = (U_{I/O} - U_B) / R_B = (5 – 0,6) / 330 = 13$ mA
$I_C = I_B * B = 13 * 750 = 10$ A (maximum)
$P_{Motor} = U_M * I_C = 12 * 10 = 120$ W (maximum)

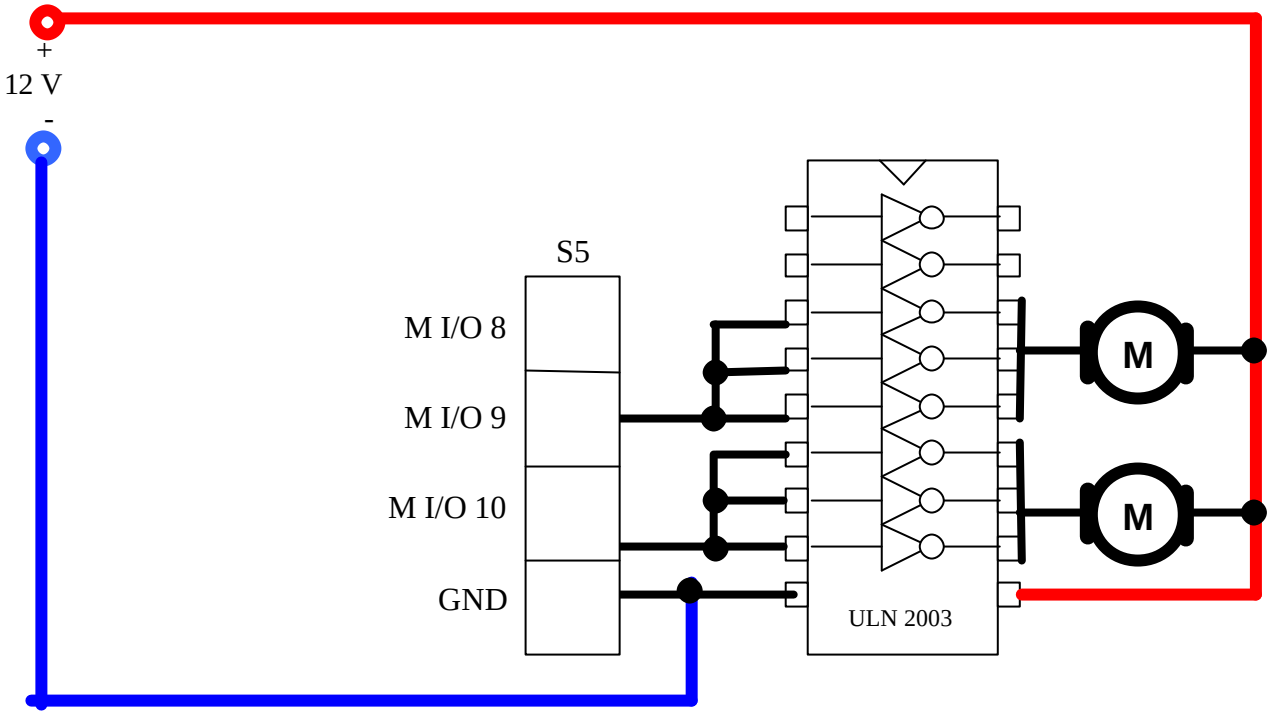
The diode, the 100 Ohm resistor and the 100 nF capacitor are employed for surge protection only.

4.3 Circuit Diagram to Control Eight Lamps



4.4 Circuit Diagram to Control Two Electrical Motors with 15 W, 12 V

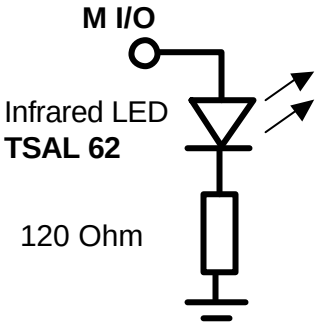
Motor current: I = 15 W / 12 V = 1,25 A
Instructions to switch the motors on/ off: HIGH, LOW and TOGGLE
The instructions PWM0 and PWM1 could supply the motors with varying voltages and control the power of the motors.



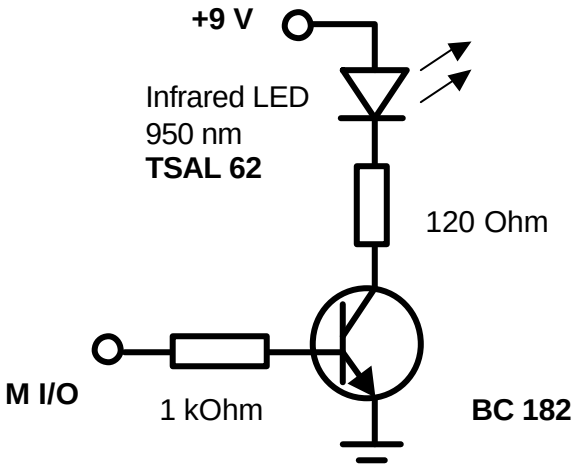
4.5 RC5 Send and Receive Circuit Diagrams

Any M I/O pin can be used for sending an RC5 code.

RCSEND pin, Word register
Distance is about 3 m.

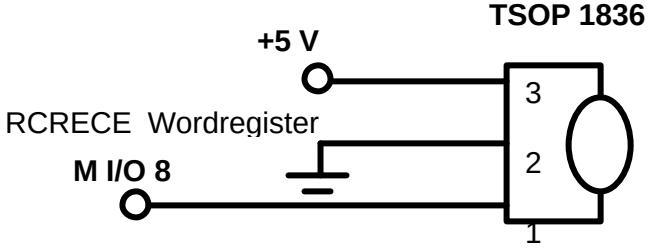


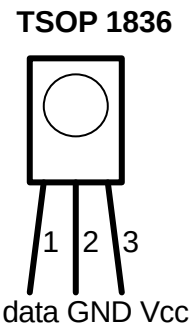
RCSEND pin , Word register
Distance is about 30 m.



RC5 receive:

infrared receiver 950 nm with filter
TSOP 1836





The buffered pin M I/O 8 on the connector S5 is used for RC5 reception.

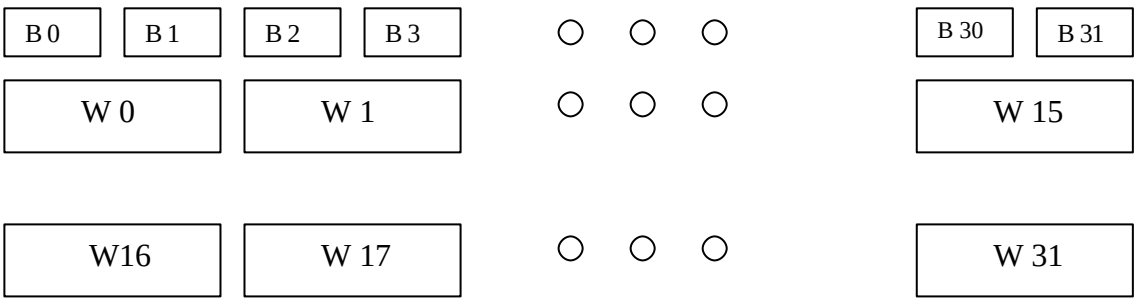
5 USB Master Hardware and Software Features

Highlights:
32 general purpose byte registers (8 bits)
32 general purpose word registers (16 bits)
16 bytes Stack
2 kB program memory
86 processor instructions
12 host commands to control, program and monitor the USB Master

5.1 General Purpose CPU Registers

The USB Master contains 64 8-bit general purpose registers accessible for programmers. The first 32 bytes are used as (Bx) byte register (B0 – B31) or (Wx) word (W0 – W15) registers, illustrated in the following Figure.
The B0 byte register is the most significant byte of the W0 word register.

The last 32 bytes can be addressed only as word registers (W16 – W31).



5.2 Special Purpose CPU Registers

Program Counter (PC): The program counter holds the 12 bit address of the current instruction being fetched from program memory (EEPROM). The PC is automatically incremented after every instruction. When a program jump occurs, the new value is automatically placed in the PC, overriding the incremented value.

Stack Pointer (SP): The stack pointer holds the address of the current top of a stack. The stack memory is organized as a last in first out (LIFO) file. Data can be pushed onto the stack or popped off of the stack through the execution of PUSH and POP instructions. The stack allows simple implementation of subroutine nesting and simplification of many types of data manipulations. The USB Master supports a 16 byte stack.

5.3 Flag Registers

Flag registers are one bit registers, they are set or reset by arithmetical and logical instructions.

Carry Flag (C): This flag is the carry from the most significant bit of the current register that completed an arithmetical or logical instruction. For example, the carry flag is set during an add instruction when a carry from the most significant bit of the result is generated.
The carry flag is also set if a borrow is generated during a subtraction instruction.
Shift instructions may also affect the carry bit.

Zero Flag (Z): This flag is set if the result of the arithmetical or logical operation is zero, otherwise it is reset.

6 USB Master Instructions

The USB Master assembler supports a variety of instructions, like:

- instructions with two parameters
- instructions with one parameter
- flow control instructions
- input/ output group or hardware instructions
- miscellaneous instructions

6.1 Instructions with Two Parameters

syntax:

[label] INSTRUCTIONSNAME par1, par2 [;comment]

The first parameter (par1) is the destination, the second parameter (par2) is the source. These parameters can be word (Wx) or byte registers (Bx) (0 <= x =< 31, 0 <= y =< 31). If the destination is a byte register, the literal value cannot exceed 255 (0xFF).
Labels and comments are not required.

ADD - adds the value of the source to the destination
states affected: carry and zero flag
syntax:
```
ADD  Wx,Wy
ADD  Wx,word value
ADD  Bx,By
ADD  Bx,byte value
```

SUB - subtracts the value of the source from the destination
states affected: carry and zero flag
syntax:
```
SUB  Wx,Wy
SUB  Wx,word value
SUB  Bx,By
SUB  Bx,byte value
```

CMP - compares the value of the source with the destination
states affected: carry and zero flag
syntax:
```
CMP  Wx,Wy
CMP  Wx,word value
CMP  Bx,By
CMP  Bx,byte value
```

OR - logical “or” operation of source and destination
state affected: zero flag
syntax:
```
OR   Wx,Wy
OR   Wx,word value
OR   Bx,By
OR   Bx,byte value
```

AND - logical “and” operation of source and destination
state affected: zero flag
syntax: AND Wx,Wy
AND Wx,word value
AND Bx,By
AND Bx,byte value

XOR - logical “exclusive or” operation of source and destination
state affected: zero flag
syntax: XOR Wx,Wy
XOR Wx,word value
XOR Bx,By
XOR Bx,byte value

MOV - loads the destination register with the value of the source
state affected: none
syntax: MOV Wx,Wy
MOV Wx,word value
MOV Bx,By
MOV Bx,byte value

Examples:

```
MOV     B1,0xFE     ;loads B1 with the value FEhex (254dec)
MOV     B2,200      ;loads B2 with the value C8hex  (200dec)
ADD     B1,B2       ;adds B1 and B2, result is C6hex (198dec) in B1,
                    ;the carry flag is set because of overflow
MOV     B5,B1       ;loads B5 with the content of B1 (C6hex)
```

6.2 Instructions with Two Word Parameters

syntax:

[label] INSTRUCTIONSNAME par1, par2 [;comments]

READ - reads the program memory (EEPROM),
Wy specifies the program memory address to be read
state affected: none
syntax: READ Wx,Wy

WRITE - writes the program memory (EEPROM),
Wy specifies the program memory address to be written
state affected: none
syntax: WRITE Wx,Wy

MUL - multiplies the two least significant bytes of the word registers
state affected: none
syntax: MUL Wx,Wy

DIV - divides the destination with the least significant byte

of the source word register
state affected: none
syntax: DIV Wx,Wy

Examples:

```
MOV     B4,0xab     ;loads literal value AB(hex) into byte register 4
MOV     B3,100      ;loads literal value 100(dec) (64(hex)) into byte register 3
ADD     B3,B4       ;adds the values in B3 and B4, result is stored in B3,
                    ;the carry is set, because 0xab + 0x64 = 0x1 1F,
                    ;an overflow occurs, the new value in B3 is 0x1F
READ    W3,W4       ;the content of the EEPROM on the address
                    ;declared by W4 is loaded into W3
WRITE   W6,W7       ;the content of W6 is written to the EEPROM
                    ;address declared in W7
```

6.3 Instructions with One Parameter

syntax:

[label] INSTRUCTIONSNAME parameter [;comments]

The parameter may be a byte or a word register. Labels and comments are not required.

INC - increments the content of a register
states affected: carry and zero flag
syntax: INC Wx
INC Bx

DEC - decrements the content of a register
states affected: carry and zero flag
syntax: DEC Wx
DEC Bx

INV - builds the 1's complement of a register
state affected: zero flag
syntax: INV Wx

CLR - clears a register
state affected: zero flag
syntax: CLR Wx

RR - rotates a register right through the carry
state affected: carry flag
syntax: RR Wx

RL - rotates a register left through the carry
state affected: carry flag
syntax: RL Wx

PUSH - pushes a register content to the stack
state affected: none
syntax: PUSH Wx

POP - gets data from the stack and stores data in a register
state affected: none
syntax: POP Wx

DEBUG - displays a register in the debug event window
state affected: none
syntax: DEBUG Wx

Examples:

```
Start:  CLR     W0          ;clears register W0
        CLR     B5          ;clears register B5
        INC     W0          ;increments register W0
        DEC     B5          ;decrements register B5
        PUSH    B5          ;pushes register B5 to the stack
        POP     B6          ;pops the current stack value (B5) to register B6
        DEBUG   B6          ;displays the value of B6 in the debug event window

        MOV     W0,0x0F0F   ;moves the binary value
                            ;0000 1111 0000 1111 to W0
        OR      B0,B0       ;clears the carry flag, C=0
        RR      W0          ;moves the binary value in W0 right:
                            ;0000 0111 1000 0111,
                            ;the least significant bit sets the carry flag, C=1

        RL      W0          ;moves the binary value in W0 left:
                            ;0000 1111 0000 1111,
                            ;the most significant bit clears the carry flag, C=0
```

6.4 Instructions with One Word Parameter

syntax:

[label] INSTRUCTIONSNAME word register [;comments]

JMPI - jumps indirect
state affected: none
syntax: JMPI Wx

DELAY - makes a pause
1000 - 15 ms
10000 - 150 ms
60000 - 0,9 s

state affected: none
syntax: DELAY Wx

TXBAUD - sets the serial interface transmit baud rate
state affected: none
syntax: TXBAUD Wx

RCRECE - receives RC5 via the pin M I/O 8,
sets the carry flag if data are successfully received,
received data is in word register,
resets the carry flag if no data is received
state affected: carry flag
syntax: RCRECE Wx

Examples:

```
        LA      W20,donothing
        JMPI    W20             ;makes a jump to label “donothing”

        MOV     W5,200          ;sets transmission baud rate to 9600 bps
        TXBAUD  W5              ;200  -   9600 Baud
                                ;400  -   4800 Baud
                                ;800  -   2400 Baud
                                ;1600 -   1200 Baud
donothing:
        PAUSE   60000           ;delay of 1 s
```

Example:

```
Try:    RCRECE  W0              ;try to receive
        JNC     Try             ;no data
        Debug   W0              ;data successfully received,
                                ;display result in debug event window
        Goto    Try
        end
```

6.5 Instructions with One Byte Parameter

syntax:

[label] INSTRUCTIONSNAME byte register [;comments]

RANDOM - generates a random value

6.6 Flow Control Instructions

syntax:

[label] INSTRUCTIONSNAME label [;comments]

6.6.1 Unconditional Jumps

JMP - jumps to an address specified by a label parameter
state affected: none
syntax: JMP label

GOTO - identical with JMP
state affected: none
syntax: GOTO label

CALL - makes a call of the subroutine specified by a label parameter
state affected: none
syntax: CALL label

RETURN - returns from a subroutine
state affected: none
syntax: RETURN

6.6.2 Conditional Jumps

JZ - jumps to the specified label if zero flag is set
state affected: none
syntax: JZ label

JNZ - jumps to the specified label if zero flag is not set
state affected: none
syntax: JNZ label

JC - jumps to the specified label if carry flag is set
state affected: none
syntax: JC label

JNC - jumps to the specified label if carry flag is not set
state affected: none
syntax: JC label

Example:

```
Start:      CALL        Subr                    ;calls the subroutine
            GOTO        Start                   ;do this loop again

Subr:                                           ;begin of the subroutine
;do something
            RETURN                              ;return from the subroutine
```

Example:

```
Start:      CLR         B8                      ;clears counter (B8)
Loop:       INC         B8                      ;B8 = B8 + 1
            JNC         Loop                    ;do while not overflow
;overflow
            echo        otx                     ;show text otx
            JMP         Start                   ;start again
otx:        STRING      “an overflow occurred !!”   ;definition of string otx
            END
```

6.7 Hardware Instructions

6.7.1 Hardware Instructions with One Word Parameter

syntax:

```
[label]     INSTRUCTIONSNAME        word register       [;comments]
```

DIRS - sets direction of M I/O 0 - 7 (and 16 – 23 *)
as specified in a word register:
1 - input
0 - output
state affected: none
syntax: DIRS Wx

PEEK - reads the state of the input pins of M I/O 0 - 7 (and 16 – 23 *) in a specified word register
state affected: none
syntax: PEEK Wx

POKE - writes the output pins of M I/O 0 - 7 (and 16 – 23 *)
as specified in a word register:
state affected: none
syntax: POKE Wx

Note: *) pins 16 - 23 are available on the USB BigMaster only.

Examples:

```
        MOV     W2,0xF0     ;binary value          0000 0000 1111 0000
                            ;inputs:                         ^^^^
                            ;outputs:              ^^^^ ^^^^      ^^^^
                            ;available pins:       FEDC BA98 7654 3210
                            ;pins used in a USB Master:      7654 3210

        DIRS    W2          ;sets pins 4 - 7 as inputs, all other pins are outputs
        PEEK    W1          ;reads all input pins, only bit 4 - 7 are valid inputs
        MOV     W0,0x3      ;binary value 0000 0000 0000 0011
        POKE    W0          ;output pins 0, 1 are set and pins 2, 3 are reset,
                            ;pins 4 - 7 are inputs and not affected
```

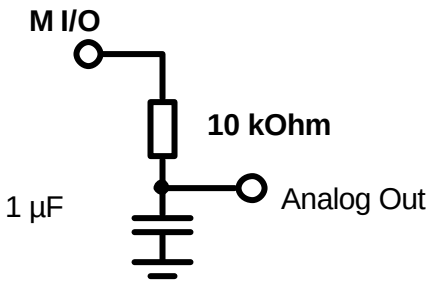
PWM0 - sets function PWM output on M I/O 10
state affected: none
syntax: PWM0 Wx

PWM1 - sets function PWM output on M I/O 9
state affected: none
syntax: PWM1 Wx

Examples:

```
;setup PWM0
        MOV     W1,0xFF80       ;Period  -  upper byte = 0xFF
                                ;value  -  lower byte = 0x80
;if upper byte < lower byte it is an error
        PWM0    W1              ;duty is about 50 % = 255 / 128

;setup PWM1
;upper byte is not used
        MOV     W2,128
        PWM1    W2              ;duty is about 50 %
```



RXINIT - initialize serial input on M I/O 15
baud rate – word register: 9600 - 155
4800 - 310
2400 - 620
1200 - 1240
state affected: none
syntax: RXINIT Wx

RXLAST - reads last received character from circular buffer, clears the buffer, sets the carry flag if character is valid
state affected: carry flag
syntax: RXLAST Wx

RXFIRST - reads first received character from circular buffer and sets the carry flag if the character is valid
state affected: carry flag
syntax: RXFIRST Wx

Examples:

```
;setup serial receiver
Start:      MOV       W3,155      ;baud rate: 9600
            RXINIT    W3
;after RXINIT, M I/O 14 and 15 are reserved for communication,
;M I/O can not be used on these pins,
;for serial transmit TX M I/O 0 is reserved on the USB Master
.
get:        RXFIRST   W2          ;get oldest character
            JC        print       ;jump if character is valid

;no character received
            RXLAST    W1          ;get newest character
            JC        print
            Goto      get

print:      DEBUG     W1          ;show character in debug event window
            Goto  get
```

USBSEND - sends the content of the specified word register via USB
state affected: none
syntax: USBSEND Wx

USBREC - reads USB telegram, sets the carry flag if telegram is received, otherwise carry flag is reset
state affected: carry flag
syntax: USBREC Wx

Examples:

```
            MOV     W1,0xABCD ;initializes W1
            USBSEND W1        ;sends user defined telegram via USB

get:        USBREC  W3        ;gets USB telegram
            JC      print     ;telegram received
;no telegram
            Goto    get
print:      DEBUG   W3        ;show character in debug event window
            Goto    get
```

ADC0 - gets value converted by ADC from input AD0
state affected: none
syntax: ADC0 Wx

ADC1 - gets value converted by ADC from input AD1
state affected: none
syntax: ADC1 Wx

Example:

```
            ADC0    W1        ;reads value on pin AD0 into lower byte
            DEBUG   B3        ;show lower byte
```

6.7.2 Hardware Instructions with One Byte Parameter

RXSTATE - gets the number of received characters in the circular buffer
state affected: none
syntax: RXSTATE Bx

PWMPRESC sets prescale value for PWM0 (allowed values 0, 1, 2, 3)
state affected: none
syntax: PWMPRESC Bx

RCFREQ sets the frequency of RCSEND:
16 - 38KHz
19 - 36 KHz
23 - 34 KHz
state affected: none
syntax: RCFREQ Bx

Example:

```
MOV       b1,0x13
RCFREQ    b1
MOV       W2,0xABC
RCSEND    5,W2          ;sends 0xabc on pin 5
...
```

Examples:

```
          MOV           B0,1
          PWMPRESC      B0          ;sets prescale for PWM0
;
          RANDOM        B5          ;loads a random value into B5
;
          RXSTATE       B6          ;loads the number of characters
                                    ;received and stored in the circular buffer
```

6.7.3 Hardware Instructions with One Pin Parameter

HIGH - set the pin specified by the parameter high (+5 V)
state affected: none
syntax: HIGH Pin

LOW - set the pin specified by the parameter low (0 V)
state affected: none
syntax: LOW Pin

TOGGLE - toggle pin – changes the state on the pin specified by the parameter
state affected: none
syntax: TOGGLE Pin

Example:

```
HIGH      0     ;sets pin M I/O 0  high (+5 V)
LOW       0     ;resets pin M I/O 0  low (0 V)
TOGGLE    1     ;toggles pin M I/O 1
```

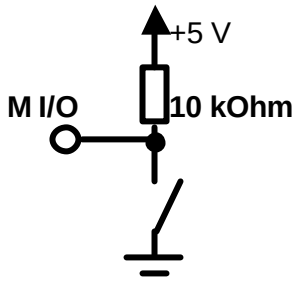
MLRESET - resets Microlan bus on the pin specified by the parameter,
sets the zero flag if Microlan client exists on the bus,
see example Microlan instructions
state affected: zero flag
syntax: MLRESET Pin

PIN - reads M I/O pin specified by the parameter, sets zero flag according to the pin state, if pin is low Z=1, else Z=0
low - 0 – 1,6 V
high - 3,4 – 5 V
state affected: zero flag
syntax: PIN Pin

Examples:

```
;read Pin 0
Start:    PIN      2          ;gets state of M I/O 2
          JZ       pinis0
;pin is high
          ECHO     pishtxt
          Goto     Start

;pin is low
pinis0:   ECHO     pisltxt
          Goto     Start
          ...
pisltxt:  STRING   "pin is low, switch is closed"
pishtxt:  STRING   "pin is high switch is not closed"
          end
```



6.7.4 Hardware Instructions with Pin and Word Parameter

syntax:

[label] INSTRUCTIONSNAME pin, word register [;comments]

The specification of labels and comments is not required.
pin - specifies the M I/O pin
word register - input or output values

SOUND - generates a peep sound on the M I/O pin specified according to the value in the word register:
- higher byte: pitch
- lower byte: time

state affected: none
syntax: SOUND Pin,Wx

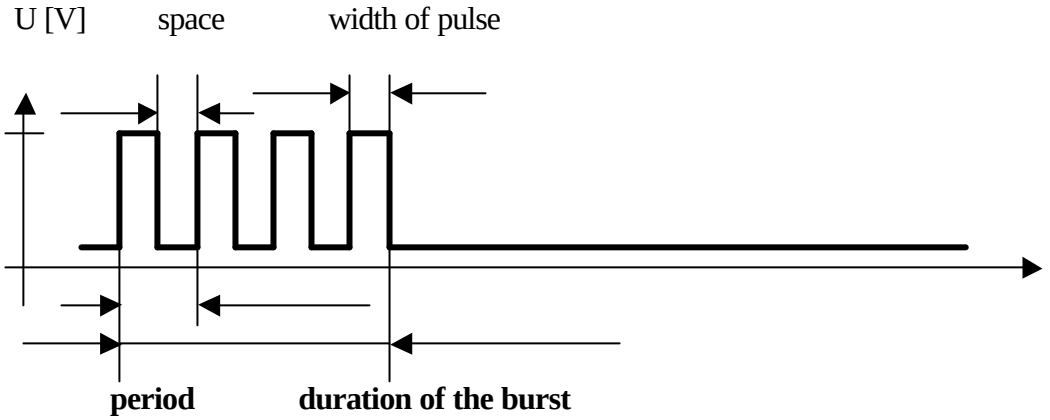
Example:

```
MOV     B0,1        ;pitch
MOV     B1,0xFF     ;time about 1s
SOUND   1,W0        ;generates sound on M I/O 1
```

Attention!
Do not connect solenoid speakers to M I/O pins. M I/O pins support only piezo speakers.

BURST - generates pulses on the specified pin
according to the value of the word register:
- higher byte: time between two pulses
- lower byte: number of pulses in the burst

state affected: none
syntax: BURST Pin,Wx



The following is defined:
- space is same length than width, period = 2 * width = 2 * space
- width = 0.8 * n [µs], n = value of the higher byte

Example:

```
MOV     B0,1        ;width = about 0,8 µs
MOV     B1,6        ;6 edges = 3 periods = 4,8 µs
BURST   1,W0        ;burst on M I/O pin 1
```

COUNT - counts the number of edges on the M I/O pin
specified by the first parameter, the duration is specified in the word parameter
state affected: none
syntax: COUNT Pin,Wx

Example

```
MOV     W2,0x8000       ;about 1/2 s
COUNT   3,W2            ;counts the edges for ½ s on pin 3
DEBUG   W2              ;show # of edges
```

PULSIN - measures the width of an electrical pulse, sets the carry flag if a pulse is detected
state affected: carry flag
syntax: PULSIN Pin,Wx

EDGE - starts with rising edge to measure the time until the falling edge, sets the carry flag if an edge is detected
state affected: carry flag
syntax: EDGE Pin,Wx

PULSOUT - generates a pulse on the specified M I/O pin
state affected: none
syntax: PULSOUT Pin,Wx

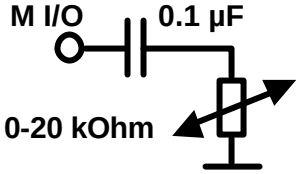
Example:

```
Start:  MOV     W1,0x1000   ;waits for 15 ms for a pulse
        PULSIN  2,W1        ;measures the width of the pulse on M I/O pin 2
        JC      pulse detected
;no pulse
        Goto    Start
pulse detected:
        Goto    Start
```

POT - measures a resistor connected to a specified M I/O pin
state affected: none
syntax: POT Pin,Wx

Example:

```
POT     3,W1        ;measures resistor
Debug   W1          :shows measured value in the debug event window
```



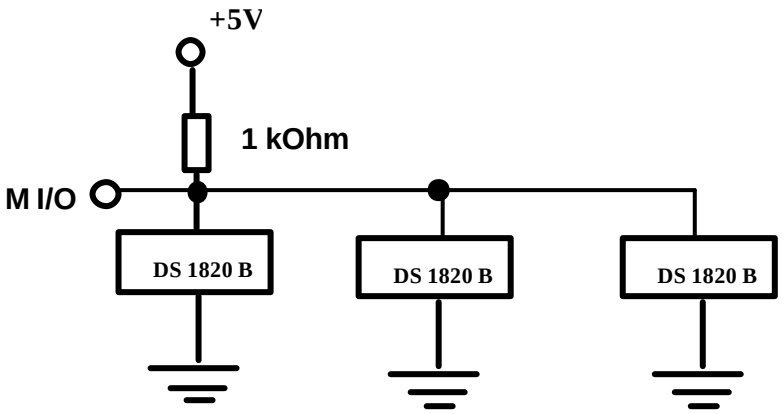
MLREAD - reads a byte from the Microlan bus
state affected: none
syntax: MLREAD Pin,Wx

MLWRITE - writes a byte to the Microlan bus
state affected: none
syntax: MLWRITE Pin,Wx

Examples:

```
;read serial number
define      MICROLAN  PIN5      ;M I/O 5 is the dedicated Microlan bus pin
define      MLVALUE   W5        ;W5 is used for Microlan transmit and receive values

;read serial number
start:
            MLRESET   MICROLAN
            JNZ       nothing
            MOV       MLVALUE,0x33
            MLWRITE   MICROLAN,MLVALUE
;1
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;2
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;3
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;4
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;5
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;6
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;7
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
;8
            MLREAD    MICROLAN,MLVALUE
            DEBUG     MLVALUE
            GOTO      start
Nothing:    echo      ntxt
            GOTO      start
Ntxt:       String    “nothing is on Microlan bus”
            End
```



RCSEND - sends an RC5 code
state affected: none
syntax: RCSEND Pin,Wx

Example:

```
MOV     W1,0x0123
RCSEND  5,W1            ;sends 0x123 on pin 5
```

SERSTR - generates an RS 232 code on the M I/O pin specified by the first parameter according to the string specified by the word register
state affected: none
syntax: SERSTR Wx

SERHEX - generates an RS 232 code on the M I/O pin specified by the first parameter according to the hexadecimal value in the word register
state affected: none
syntax: SERHEX Wx

SERDEC - generates an RS 232 code on the M I/O pin specified by the first parameter according to the decimal value in the word register
state affected: none
syntax: SERDEC Wx

SEROUT - generates an RS 232 code on the M I/O pin specified by the first parameter according to the value in the lower byte of the word register
state affected: none
syntax: SEROUT Wx

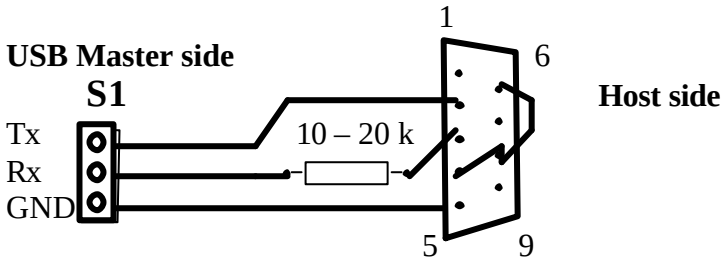
Examples:

Start:

```
MOV     W0,200          ;200=9600 Baud, 400=4800 Baud
TXBAUD  W0              ;sets transmit baud rate to 9600
LA      W1,hs           ;loads start address of string hs
SERSTR  0,W1    :       ;transmits ”Hello World!!” on TX (S1), M I/O 0
```

```
            MOV     W0,100          ;loads 100DEC = 64HEX
            SERDEC  0,W0            ;transmits on TX "100"

            MOV     W0,0x41         ;loads "A"
            SEROUT  0,W0            ;transmits "A" on TX
            GOTO    start
hs:         String  "Hello World!!"
            END
```



RS 232 Cable

6.8 Miscellaneous

SETUP - writes the SFR/ GPR specified by the lower part of the value with the content specified by the higher part of the value
state affected: none
syntax: SETUP Value

SFR = special purpose register
GPR = general purpose register

Example:

```
            SETUP   0x1203          ;writes content 12 onto address 3
                                    ;FSR has the address 3
                                    ;see PIC 16C7x5 manual for details
```

ENABLE - enables interrupt as specified by MASK
state affected: none
syntax: ENABLE Mask

DISABLE - disables all interrupts
state affected: none
syntax: DISABLE

RETI - returns from interrupt subroutine
state affected: none
syntax: RETI

Examples:

```
                DISABLE                 ;clears all interrupts
                ENABLE      0x0C        ;enables interrupts 2,3
                                        :0x0C  =  0000 1100
idle:
                echo        n
                delay       10000
                toggle      1           ;show the loop through a LED on pin 1
                jmp         idle

i0:                                     ;ISR 0 disabled
                inc         B0
                debug       B0
                echo        s0
                reti
i1:                                     ;ISR 1 disabled
                inc         B1
                debug       B1
                echo        s1
                reti
i2:                                     ;ISR 2 enabled
                inc         b2
                debug       b2
                echo        s2
                reti
i3:                                     ;ISR 3 enabled
                inc         b3
                debug       b3
                echo        s3
                reti

;this ISR happens only if an error occurs
eri:
                echo        ert
                reti

;string definitions

ert:            string      "Interrupt Error"
s0:             string      "int0"
s1:             string      "int1"
s2:             string      "int2"
s3:             string      "int3"
n:              string      "nix"
```

```
;interrupt table
        org     0x100           ;start address: hex 100

        da      i0              ; insert address of ISR 0
        da      i1              ; insert address of ISR 1
        da      i2              ; insert address of ISR 2
        da      i3              ; insert address of ISR 3
        da      eri             ; insert address of ISR 4 /error
        da      eri             ; insert address of ISR 5 /error
        da      eri             ; insert address of ISR 6 /error
        da      eri             ; insert address of ISR 7 /error
        end
```

ECHO - displays a string in the events debug window
state affected: none
syntax: ECHO Label

Example:

```
        ECHO    string1

;string definition
string1:    STRING  “Hello”
```

BREAK - stops the CPU - use the instruction only for test purposes !
state affected: none
syntax: BREAK

Example:

```
Start:  CLR     B0
Loop:
        INC     B0
        DEBUG   B0
        BREAK               ;the CPU stops unconditionally on this instruction
        Jmp     Loop
```

LA - loads address of a label
state affected: none
syntax: LA Wx,Label

Example:

```
Start:  LA      W0,abc      ;W0=0x20 address of abc string
        DEBUG   W0

        ORG  0x20  ...
abc:    STRING     "ABC"
```

STATE - reads SFR/ GPR specified by the address
state affected: none
syntax: STATE Wx,Value

Example:

```
        STATE   W0,0x3000   ;reads byte on address 0x30 into W0 (B0),
                            ;higher part of 2nd parameter is a valid address of the
                            RAM, the lower part is not used
```

6.9 Assembler Directives

Assembler directives control the functions of the Assembler, they do not generate code for the CPU.

ORG - organizes address
state affected: none
syntax: ORG Value

STRING - defines string
state affected: none
syntax: STRING "content of string"

DB - defines bytes
state affected: none
syntax: DB Value1,Value2,...

DA - defines addresses, see interrupt example
state affected: none
syntax: DA label,...

DW - defines words
state affected: none
syntax: DW Value1,Value2,...

END - end of compilation
state affected: none
syntax: END

Examples:

```
Start:    ECHO    str
          Jmp     Start
Str:      STRING  “This is a string”
Bytes:    db      1,2,3,4,0xa,0xb          ;defines content of bytes
Words:    dw      0xabcd, 0xFFFF           ;defines content of words
          End
```

6.10 Host Control Commands

The following commands allow a PC (host) to control the USB Master (target) via a USB interface. The software on the PC has to have an 8 byte input and output buffer. The control commands make USB Master remote functions like download and debug available to the PC.

USB Vendor ID of the USB master: 04 61 hexadecimal
Product ID: 00 65 hexadecimal

The bytes of the buffer not commented are not be used and don’t matter.

Break - stops the virtual CPU on the target
Command: Byte 0 = 0
Echo: general state of the USB Master see USB Master reports

Go - lets the target run from current address
Command: Byte 0 = 1
Echo none

Single step - lets the target do a single instruction
Command: Byte 0 = 2
Echo: general state of USB Master after the single step

Get octad - target sends 8 bytes of the SFR (Special Purpose Register) or GPR (General Purpose Register), the start address is specified in byte 1
Command: Byte 0 = 3
Byte 1 = 0 – 0x3f number of octets
Echo 8 Bytes

Write RAM/SFR - target overwrites a byte on the memory location specified in byte 1, 2
Command: Byte 0 = 4
Byte 1 = lower address
Byte 2 = higher address
Byte 3 = byte to written
Echo: none

Read RAM/SFR - target reads a byte from memory location specified in byte 1, 2

Command:	Byte 0	=	5
	Byte 1	=	lower address
	Byte 2	=	higher address
Echo	Byte 0	=	byte read

Write PROM - target writes a byte on external EEPROM location specified in byte 1, 2

Command:	Byte 0	=	6
	Byte 1	=	lower address
	Byte 2	=	higher address
	Byte 3	=	byte to written
Echo	none		

Read PROM - target reads a byte on external EEPROM location specified in byte 1, 2

Command:	Byte 0	=	7
	Byte 1	=	lower address
	Byte 2	=	higher address
Echo	none		

Version - target sends the major, minor and dotversion number

Command:	Byte 0	=	8
Echo	Byte 0	=	major version +0x80
	Byte 1	=	minor version
	Byte 2 – dotversion		

Read 8 bytes of ROM

target sends content of 8 bytes, the start address is specified in byte 1, 2

Command:	Byte 0	=	9
	Byte 1	=	address low
	Byte 2	=	address high
Echo	8 Bytes		
	start address is 1. command byte * 8		

USB user received

user program receives a word from host

Command:	Byte 0	=	0xA
	Byte 1	=	higher part of word
	Byte 2	=	lower part of word
Echo	none		

6.11 USB Master Reports

USB Master (target) sends the following reports to the PC (host) via the USB.

Break - target reaches a BREAK instruction, stops, sends state

Byte 0	=	0
Byte1	=	lower address
Byte2	=	higher address

Show Byte - user program reports a byte register specified in byte 1

Byte0	=	1
Byte1	=	number of the byte
Byte2	=	content of the byte

Show Word - user program reports a word register specified in byte 1

.

Byte0	=	2
Byte1	=	number of words
Byte2	=	lower part of content
Byte3	=	higher part of content

Show String
- user program reports 7 characters of a string,
- if the string is shorter than 7 characters it fills the remaining fields with 0s

Byte0	=	3
Byte1	=	first character of the string
Byte2	=	second character of the string
...		
Byte 7	=	7.th character of the string

Unknown command

- user program reached a not specified instruction in program field

Byte0	=	4
Byte1	address low	
Byte2	address high	

Zero divide error

Byte0	=	5
Byte1	=	address low
Byte2	=	address high

User defined - user program sends 7 bytes

Byte0	=	6
Byte1 – 7	=	user defined

General State after single step or break

The USB Master sends after a break and single step command the content of 7 registers with the start address.

Byte0	bit 0 – bit 6 physical address of the first register
	Bit 7 - 1
Byte1	1. register
Byte2	2. register
Byte3	3. register
Byte4	4. register
Byte5	5. register
Byte6	6. register
Byte7	7. register

Remarks:
Address 0 represents the user register B0 or the most significant byte of W0. The user register space is 0 – 0x13 (64 registers) and the stack.
Address 5 reports the port information of PIC 16C7x5
Address 5 reports the TRIS information of PIC 16C7x5
Address 0x40 reports the state of the virtual CPU

0x40	higher byte of the Program Counter (PC)
0x41	lower byte of the PC
0x42	stack pointer
0x42	intern state
0x43	hardware state
0x44	interrupt mask
0x45	interrupt serviced

7 USB Master Processor Instructions and Machine Code

The following processor commands are interpreted by the USB Master. They are fetched from the EEPROM.

Abbreviations:
- d bit of destination specification
- S bit of source specification
- P bit of pin specification
- A bit of address specification
- x bit of value
- - don't care

Byte literal

	Code	1. Par	2. Par
Movbl	00	000 d dddd	xxxxxxxx
Addbl	00	001 d dddd	xxxxxxxx
Subbl	00	010 d dddd	xxxxxxxx
Cmpbl	00	011 d dddd	xxxxxxxx
Andbl	00	100 d dddd	xxxxxxxx
Orbl	00	101 d dddd	xxxxxxxx
Xorbl	00	110 d dddd	xxxxxxxx
enable	00	111 0 0001	- MASK -
undef	00	111 0 0010	xxxxxxxx

Byte - byte

	Code	1. Par	2. Par
movb	10	0000 00 dd	ddd s ssss
addb	10	0000 01 dd	ddd s ssss
subb	10	0000 10 dd	ddd s ssss
cmpb	10	0000 11 dd	ddd s ssss
andb	10	0001 00 dd	ddd s ssss
orb	10	0001 01 dd	ddd s ssss
xorb	10	0001 10 dd	ddd s ssss
read ram	10	0001 11 dd	ddd s ssss
write ram	10	0010 00 dd	ddd s ssss
undef	10	0010 01 dd	ddd s ssss
undef	10	0010 10 dd	ddd s ssss
undef	10	0010 11 dd	ddd s ssss

Word – word

	Code	1. Par		2. Par	
Movw	10	0100	00 dd	ddd s	ssss
addw	10	0100	01 dd	ddd s	ssss
subw	10	0100	10 dd	ddd s	ssss
cmpw	10	0100	11 dd	ddd s	ssss
andw	10	0101	00 dd	ddd s	ssss
orw	10	0101	01 dd	ddd s	ssss
xorw	10	0101	10 dd	ddd s	ssss
mulw	10	0101	11 dd	ddd s	ssss
divw	10	0110	00 dd	ddd s	ssss
readrom	10	0110	01 dd	ddd s	ssss
writerom	10	0110	10 dd	ddd s	ssss
undefined	10	0110	11 dd	ddd s	ssss
undefined	10	0111	00 dd	ddd s	ssss

Hardware **commands pin and wordpar**

	Code	1. Par		2. Par	
Sound	10	1000	00 pp	ppp d	dddd
Burst	10	1000	01 pp	ppp d	dddd
Count	10	1000	10 pp	ppp d	dddd
Pulsin	10	1000	11 pp	ppp d	dddd
;pwm	10	1001	00 pp	ppp d	dddd
Pulsout	10	1001	01 pp	ppp d	dddd
Pot	10	1001	10 pp	ppp d	dddd
Mlread	10	1001	11 pp	ppp d	dddd
Mlwrite	10	1010	00 pp	ppp d	dddd
;undefined					
rcsend	10	1010	10 pp	ppp d	dddd
Serstr	10	1010	11 pp	ppp d	dddd
Serhex	10	1011	00 pp	ppp d	dddd
Serdec	10	1011	01 pp	ppp d	dddd
serout	10	1011	10 pp	ppp d	dddd
edge	10	1011	11 pp	ppp d	dddd

Word literal

	Code	1. Par		2. Par	3. Par
movwl,la	20	000 d	dddd	xxxxxxxx	xxxxxxxx
addwl	20	001 d	dddd	xxxxxxxx	xxxxxxxx
subwl	20	010 d	dddd	xxxxxxxx	xxxxxxxx
cmpwl	20	011 d	dddd	xxxxxxxx	xxxxxxxx
andwl	20	100 d	dddd	xxxxxxxx	xxxxxxxx
orwl	20	101 d	dddd	xxxxxxxx	xxxxxxxx
xorwl	20	110 d	dddd	xxxxxxxx	xxxxxxxx
					<- TYP 3
mulwl	20	111 d	dddd	xxxxxxxx	xxxxxxxx
divwl	30	000 d	dddd	xxxxxxxx	xxxxxxxx
					<- TYP 6
setup	30	001 -	----	AAAAAAAA	-value--
state	30	010 -	value	AAAAAAAA	xxxxxxxx
delay	30	011 -	----	xxxxxxxx	xxxxxxxx
undef	30	100 d	dddd	xxxxxxxx	xxxxxxxx

Byte commands

	Code	1. Par	
Incb	40	000 d	dddd
decb	40	001 d	dddd
invb	40	010 d	dddd
clrb	40	011 d	dddd
rrb	40	100 d	dddd
rlb	40	101 d	dddd
pushb	40	110 d	dddd
popb	40	111 d	dddd
debugb	50	000 d	dddd
randomb	50	001 d	dddd
rxstate	50	010 d	dddd
pwmpresc	50	011 d	dddd
rc5freq	50	100 d	dddd
;usbput	50	101 d	dddd
;rcrece	50	110 d	dddd
;rcsend	50	111 d	dddd

Word commands

	Code	1. Par
incw	60	000 d dddd
decw	60	001 d dddd
invw	60	010 d dddd
clrw	60	011 d dddd
rrw	60	100 d dddd
rlw	60	101 d dddd
pushw	60	110 d dddd
popw	60	111 d dddd
debugw	70	000 d dddd
Jmpi	70	001 d dddd
Pause	70	010 d dddd
txbaud	70	011 d dddd
rcrece	70	100 d dddd

Hardware commands with word arguments

	Code	1. Par
Dirs	80	000 d dddd
Peek	80	001 d dddd
Poke	80	010 d dddd
Hwpwm,pwm0	80	011 d dddd
Pwm1	80	100 d dddd
rxinit	80	101 d dddd
rxfirst	80	110 d dddd
rxlast	80	111 d dddd
usbsend	90	000 d dddd
usbrec	90	001 d dddd
adc0	90	100 d dddd
adc1	90	101 d dddd
adc2	90	110 d dddd
undef	90	111 d dddd

Hardware commands with pin arguments

	Code	1. Par
High	a0	000 p pppp
Low	a0	001 p pppp
toggle	a0	010 p pppp
In	a0	011 p pppp
mlreset	a0	100 p pppp
Pin	a0	101 p pppp
undef	a0	110 p pppp
undef	a0	111 p pppp

Jumps

```
Jmp       1011 0AAA AAAA AAAA
Call      1100 0AAA AAAA AAAA
Debugs    1101 0AAA AAAA AAAA
Jz        1110 0AAA AAAA AAAA
Jnz       1111 0AAA AAAA AAAA
Jc        0000 1AAA AAAA AAAA
Jnc       0001 1AAA AAAA AAAA
Undef     0010 1xxx xxxx xxxx
```

Miscellaneous commands without parameter

```
Break     0011 1000
Ret       0011 1001
Disable   0011 1010
Reti      0011 1011
```

Directives

```
Org
String
Db
Da
Dw
End
```