SECONS

MicroVGA conio/text user interface library and demo Manual

WWW: http://www.MicroVGA.com/

Table of contents

Library license (GNU GPL)

Standard conio routines

These routines are hardware, architecture and compiler independent.

Output functions

_cputs

Puts a string to the MicroVGA.

```
void _cputs(const char *str);
```

Parameters

str	Output string

Description

The **_cputs** function writes the null-terminated string pointed to by str to the MicroVGA. A carriage return–line feed (CR-LF) combination is not automatically appended to the string.

Requirements

Required header: <conio.h>

---

textattr

Sets both the foreground and the background colors in a single call.

```
void textattr(int attr);
```

Parameters

attr	Encodes color information: 7 6 5 4 3 2 1 0 B b b b f f f f In this 8-bit newattr parameter: • ffff = 4-bit foreground color (0 to 15). • bbb = 3-bit background color (0 to 7). • B = blink-enable bit.

Description

This function does not affect any characters currently shown on the screen. Once you have called this function, all subsequent functions using direct video output (such as **_cputs**) will use the new attributes or colors.

If you use Symbolic color constant, the following limitations apply to the background colors you select:

- You can only select one of the first eight colors (0--7).
- With textattr, you must shift the selected background color to the left by 4 bits to move it into the correct "bbb" bit positions.

Requirements

Required header: <conio.h>

---

textbackground

Selects a new text background color.

```
void textbackground(int color);
```

Parameters

color	Encodes background color: 7 6 5 4 3 2 1 0 x x x x B b b b In this 8-bit newattr parameter: • xxxx = 4-bit unused. • bbb = 3-bit background color (0 to 7). • B = blink-enable bit.

Description

This function does not affect any characters currently shown on the screen. Once you have called this function, all subsequent functions using direct video output (such as **_cputs**) will use the new attributes or colors.

If you use symbolic color constants the following limitations apply to the background colors you select:

- You can only select one of the first eight colors (0--7).

Requirements

Required header: <conio.h>

---

textcolor

Selects the foreground character color.

```
void textcolor(int color);
```

Parameters

color	Encodes foreground color: 7 6 5 4 3 2 1 0 x x x x f f f f In this 8-bit newattr parameter: • xxxx = 4-bit unused. • ffff = 4-bit foreground color (0 to 15).

Description

This function does not affect any characters currently shown on the screen. Once you have called this function, all subsequent functions using direct video output (such as **_cputs**) will use the new attributes or colors.

You can use symbolic color constants to specify foreground color.

Requirements

Required header: <conio.h>

---

clrscr

Clears the screen.

```
void clrscr(void);
```

Description

Clears the current screen and places the cursor in the upper left-hand corner (at position 1,1).

Requirements

Required header: <conio.h>

---

crleol

Clears characters to the end of line on the screen.

```
void clreol(void);
```

Description

Clears all characters from the cursor position to the end of the line, without moving the cursor. Current background color is applied.

Requirements

Required header: <conio.h>

---

gotoxy

Positions cursor on the screen.

```
void gotoxy(char x, char y);
```

Parameters

x	Horizontal position relative to the origin of the screen (upper-left corner [1,1]). Maximum value is 80.
y	Vertical position relative to the origin of the screen (upper-left corner [1,1]). Maximum value is 25.

Description

Moves the cursor to the given position on the screen. If the coordinates are invalid (out of screen), the call to gotoxy is ignored.

Requirements

Required header: <conio.h>

---

cursoron

Displays the cursor.

```
void cursoron(void);
```

Description

Switches the cursor on at current position.

Requirements

Required header: <conio.h>

---

cursoroff

Hides the cursor.

```
void cursoroff(void);
```

Description

Switches the cursor off (i.e. the cursor is no longer visible).

Requirements

Required header: <conio.h>

Input functions

_cgets

Gets a character string from the MicroVGA.

```
char * _cgets(char *s);
```

Parameters

s	Storage location for data.

Return value

The **_cgets** function return a pointer to the start of the string, at buffer[2].

Description

The **_cputs** function reads a string of characters from the MicroVGA and stores the string and its length in the location pointed to by s. The s parameter must be a pointer to a character array. The first element of the array, s[0], must contain the maximum length (in characters) of the string to be read. The array must contain enough elements to hold the string, a terminating null character ('\0'), and 2 additional bytes. The function reads characters until a carriage return–line feed (CR-LF) combination or the specified number of characters is read. The string is stored starting at s[2]. The function then stores the actual length of the string in the second array element, s[1]. If escape key is pressed, getting string is canceled and zero length string is returned (s[1] is equal to 0).

Requirements

Required header: <conio.h>

Low-level hardware routines

These hardware dependent functions must be implemented by the user.

_getch

Gets a character from the MicroVGA without echo.

```
int _getch (void);
```

Return value

Returns the character read. There is no error return.

Description

The **_getch** function reads a single character or extended character (function key, arrow key) from the MicroVGA without echoing the character. Extended character is two byte long. Single character is one byte long.

Implementation

UART is used to read keyboard input from MicroVGA. Reading single ASCII character is very simple, extended character such as F-keys or cursor movement is read in two steps. First byte read

from MicroVGA, is zero, then next byte contains key code. If function key code is read, it's value is added or-ed with 0x100 in order to distinguish from printable ASCII codes. For example key F4 is read as two bytes 0x00 and then 0x3E, but function returns code 0x013E.

Example of implementation

```
int _getch (void)
{
  int ch;

  //wait until data is received
  while(!isNewDataAvailable());

  //read first byte from UART
  UART_ByteReceive(&ch);

  if (ch == 0)
  { //extended key pressed, read second byte
      //wait until data is received
     while(!isNewDataAvailable());
     UART_ByteReceive(&ch);
     //Set first byte to one
      ch = 0x100 | ch;
  }
  return ch;
}
```

Requirements

Required header: <conio.h>

_cputch

Writes a character to the MicroVGA.

```
void _putch (char ch);
```

Parameters

ch	Character to be output.

Description

The **_cputch** function writes single character **ch** to the MicroVGA.

Implementation

MCU UART module is usually used to write data to MicroVGA.

Example of implementation

```
void _putch (char ch)
{
  //Wait uVGA to be ready to receive data
  //low -> uVGA is ready to recv data
  while (BitRead(CTS_PIN_NUMBER));

  //Write byte to UART
  UART_ByteSend(&ch);

  //wait until the data transmission is finished
   while(!uart_write_finished()) ;
 }
```

Requirements

Required header: <conio.h>

_kbhit

Checks the MicroVGA for keyboard input.

```
int _kbhit (void);
```

Return value

The **_kbhit** function returns a nonzero value if a key has been pressed. Otherwise, it returns 0.

Description

The **_kbhit** function checks the MicroVGA for a recent keystroke. If the function returns a nonzero value, a keystroke is waiting in the buffer. The program can then call **_getch** to get the keystroke.

Implementation

UART is used to read/write data from/to MicroVGA. A given UART state register should be read to detect keypress.

Requirements

Required header: <conio.h>

Symbolic color constants

Constant	Value	Background	Foreground
BLACK	0	Yes	Yes
RED	1	Yes	Yes

GREEN	2	Yes	Yes
BROWN	3	Yes	Yes
BLUE	4	Yes	Yes
MAGENTA	5	Yes	Yes
CYAN	6	Yes	Yes
LIGHTGRAY	7	Yes	Yes
DARKGRAY	8	No	Yes
LIGHTRED	9	No	Yes
LIGHTGREEN	10	No	Yes
YELLOW	11	No	Yes
LIGHTBLUE	12	No	Yes
LIGHTMAGENTA	13	No	Yes
LIGHTCYAN	14	No	Yes
WHITE	15	No	Yes
BLINK	128	No	---

User interface functions

drawfkeys

Display Northon Commander-style function key bar on bottom of the screen.

```
void drawfkeys(const char *fkeys[]);
```

Parameters

fkeys	Array of strings to display in the menu.

Description

The **drawfkeys** iterate through **fkeys** array of string and display every string on the bottom of the screen. Displayed menu looks like Northon Commander function key menu. Input array of strings **(fkeys)** must contain ten (10) elements. String printing is skipped, if zero is specified in the place of string in **fkeys** array. Next string is printed in a next position.

Example

```
const static char *memview_fkeys[10] =
{"Help",0,0,0,"DynMem",0,0,0,0,"Quit"};
drawfkeys(memview_fkeys);
```

Requirements

Required header: <ui.h>

---

runmenu

Displays text-based selection menu on the screen.

```
int runmenu(char x, char y, const char *menu[], int defaultitem);
```

Parameters

x	Horizontal position of upper-left corner of menu frame. Position is relative to the origin of the screen (upper-left corner [1,1]). Maximum value is 80.
y	Vertical position of upper-left corner of menu frame. Position is relative to the origin of the screen (upper-left corner [1,1]). Maximum value is 25.
menu	Array of strings to display in the menu. Last item must be zero (0), to indicate the end of array.
defaultitem	Item of the menu, that will be highlighted as selected.

Return value

The **runmenu** function return index of item, that was selected, when enter key was pressed.

Description

The **runmenu** function displays text-based menu, that allows the user, to select desired item.

Selection is made by pressing down/up arrow key. The **runmenu** function terminates and returns the selected item index, when enter key is pressed. This index is then used by application to run desired code, that is dependent on menu selection.

The height of the menu frame is defined by number of items in the **menu** array of strings. The width is defined by longest string in the **menu** array.

Example

```
const static char *mainmenu[] =
{
  "Memory viewer",
  "Console debug demo",
  "Speed test",
  "Test sub menu",
  "Command line interface",
  "ASCII-Art test",
  "Input dialog test",
  0
};

//...
while (1) {
      item = runmenu(5,5, mainmenu, item);
      switch (item) {
        case 1:  memviewer(0); break;
        case 2:  debugdemo(); break;
        case 3:  speedtest(); break;
        case 4:  submenutest(); break;
        case 5:  cli(); break;
        case 6:  ascii_art(); break;
        case 7:  testCGetS(); break;
      }
```

Requirements

Required header: <ui.h>

---

drawframe

Display empty frame on the screen.

```
void drawframe(int x, int y, int width, int height, int color);
```

Parameters

x	Horizontal position of upper-left corner of the frame. Position is relative to the

	origin of the screen (upper-left corner [1,1]). Maximum value is 80.
y	Vertical position of upper-left corner of the frame. Position is relative to the origin of the screen (upper-left corner [1,1]). Maximum value is 25.
width	Length of text, that can be printed into the frame.
height	Number of lines, that can be printed into the frame.
color	Border (text) and background color of the frame. Refer to textattr function for color specification.

Description

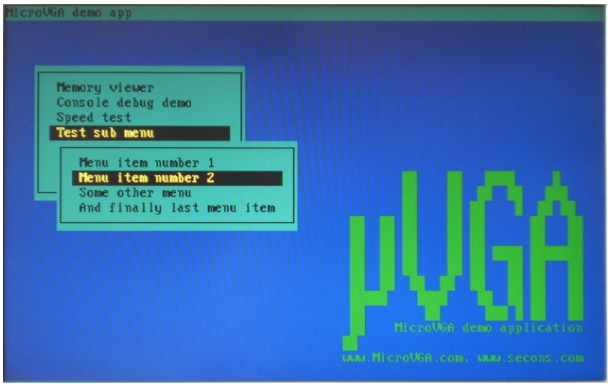
The **drawframe** function displays empty frame in the specified position on the screen. Border of the frame is created from box drawing characters hence the color parameter alter background and border/text color same as the textattr function. Width parameter specifies length of text, that can be printed into the frame, thus absolute width of the frame is four (4) characters bigger. (One [1] border character and one [1] space on the left/right sides). The height parameter specifies number of lines, that can be printed into the frame, thus the absolute height of the frame is four (4) characters bigger. (One [1] border character and one [1] space on the top/bottom sides).

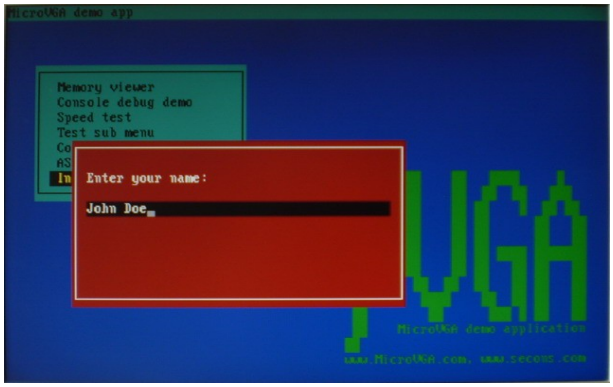
Requirements

Required header: <ui.h>

Demo Application

MicroVGA-conio demo application contains everything needed to begin creating user friendly embedded applications, using low-cost 8bit microcontrollers or feature-rich 32-bit SoC. This demo application fits even into 4kByte 8bit microcontroller!





Directory contents

Root directory of demo application consists of the following files and directories:

Name	Type	Description
h	dir	Contains header files of conio MicroVGA library(Common for every MCU).
Lib	dir	Contains C files of conio MicroVGA library(Common for every MCU).
pic24_mplab	dir	Contains MPLab project for PIC24 MCU. This project implements specific hardware routines (_kbhit, _cgets, _cputs) for PIC24. These routines are defined using UART including handshaking.
str712_iar	dir	Contains IAR project for STR712 MCU. This project implements hardware specific routines (_kbhit, _cgets, _cputs) for STR712. These routines are defined using UART0 including handshaking.
nec78k0s_iar	dir	Contains IAR project for NEC 78k0 8bit MCUs
lpc21xx_iar	dir	Contains IAR project for NXP LPC2103/04/05/06
tms470_iar	dir	Contains IAR project for Texas Instruments TMS470R1Axxx
atmega128_avrstudio	dir	Contains AVR Studio project for ATmega MCUs. This project implements hardware specific routines (_kbhit, _cgets, _cputs) for Atmega103 (single-

		uart) or Atmega128 (dual-uart). These routines are using UART with flow-control.
main.c	file	Contains main function, that is displaying the demo application. (Common for every MCU)
aa.c	file	ASCII art demo

General description

The main Demo function calls **MCU_Init** function**,** which must be defined for every MCU project. **MCU_Init** function should be used to initialize specific MCU (setting MCU clock frequency, UART initialization and settings, ...). Thus this function is defined in MCU project as well as hardware specific routines (_kbhit, _cgetch, _cputch). The main Demo function starts calling library function to display the demo after **MCU_Init** call.

STR712 MCU project

Root STR712 MCU project directory contains following files and directories:

Name	Type	Description
include	dir	Contains header files for ST library configuration and interrupt configuration.
library	dir	Directory where STR71x Firmware Library should be downloaded to
linker	dir	Contains linker files
startup	dir	ASM for initialization of ST MCU
71x_it.c	file	Interrupt service routines. Included only for linking (all service routines are empty).
hw_st710.c	file	Using STR71x Firmware Library to implement hardware specific function (**_kbhit**, **_cputch**, **_cgetch** and **Init_MCU**).

Project was created using IAR Embedded Workbench IDE V5.4.

Project is using UART0 for communication. This can be easily changed by **Use_UARTx** define in hw_st710.c. Pins P1_4 and P1_5 are used for UART handshaking. Pins numbers are set by **RTS_PIN_NUMBER** and **CTS_PIN_NUMBER** define. UART communication speed (baudrate), can be set using ST library function **UART_Config**. Note that UART baudrate depends on PCLK1 frequency. Real baudrate is subject of division and rounding and can be different from value, that is passed to **UART_Config**. (Refer to STR71xF reference manual).

STR71x Firmware Library is not included in Demo Application (license reasons), but can be downloaded at http://www.st.com/stonline/products/support/micro/files/str71xfwlib.zip for free.

Pin	Direction	Connect to uVGA	Description
P1.4	OUT	RTS	Pin is set (written) high to stop MicroVGA sending data. Pin is set (written) low to allow MicroVGA sending data.
P1.5	IN	CTS	Pin is read to find out, if MicroVGA is able to receive data. High means that MicroVGA is busy, low means ready to receive.
P0.8	IN	TXD	Standard UART receive pin.
P0.9	OUT	RXD	Standard UART transmit pin.

PIC24 MCU project description

Root PIC24 MCU project directory consists of following files and directories:

Name	**Type**	**Description**
hw_pic24.c	file	Implementing specific hardware functions (**_kbhit**, **_cputch**, **_cgetch** and **Init_MCU**)
p24HJ32GP202.h	file	PIC24 MCU header file (register memory and peripheral devices definitions).

Oscillator: internal

NEC78k0s MCU

Root NEC78k0s MCU project directory contains the following files and directories:

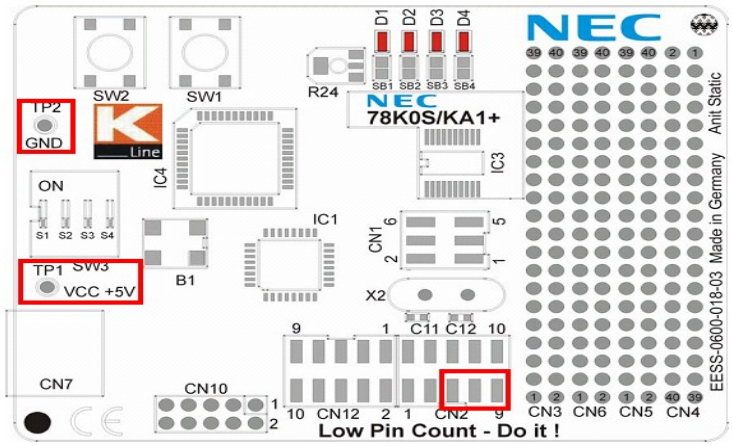
Name	**Type**	**Description**
hw_78k0s.c	file	Implementing specific hardware functions (**_kbhit**, **_cputch**, **_cgetch** and **Init_MCU**)
DF9222_V4.XCL	file	Linker file for NEC78k0s.

Project was created using IAR Embedded Workbench Kickstart for NEC 78k V4.2, which is shipped with NEC Low Pin Count Do It Starter Kit (http://www.eu.necel.com/products/micro/065_dev_tools/030_starterkits/010_starterkits_78K0S/index.html). Low-pin-count Do it! Demonstration board was used as a host hardware. (schematic diagram is available at http://www.eu.necel.com/_pdf/U18139EE1V0UM00.PDF).

Please note that NEC78k0s has only 4kB of memory, which limits the demonstration application. (Some features were removed for use with this MCU).

Pin	**Dir**	**Connect to uVGA**	**Description**
P20 (CN12.5)	IN	CTS	Pin is read to find out, if MicroVGA is able to receive data. High means that MicroVGA is busy, low means ready to receive.
P44_RxD6 (CN12.7)	IN	TXD	Standard UART receive pin.
P43_TxD6 (CN12.9)	OUT	RXD	Standard UART transmit pin.

Connect MicroVGA to the LPC-DOIT board using red-marked pins:



LPC21xx MCU project

Root LPC21xx MCU project directory consists of following files and directories:

Name	**Type**	**Description**
hw_lpc21xx.c	file	Implementing specific hardware functions (**_kbhit**, **_cputch**, **_cgetch** and **Init_MCU**)
ma_sfr.h	file	Special function register bitfield macros.

Project was created using IAR Embedded Workbench IDE V5.4. Phillips LPC2104 MCU was used. More information about this MCU can be found at http://www.nxp.com/#/pip/pip=[pip=LPC2104_2105_2106_7]|
pp=[t=pip,i=LPC2104_2105_2106_7].

Pin	**Dir**	**Connect to uVGA**	**Description**
P0.3	OUT	RTS	Pin is set (written) high to stop MicroVGA sending data. Pin is set (written) low to allow MicroVGA sending data.
P0.2	IN	CTS	Pin is read to find out, if MicroVGA is able to receive data. High means that MicroVGA is busy, low means ready to receive.
P0.1/RXD0	IN	TXD	Standard UART receive pin.
P0.0/TXD0	OUT	RXD	Standard UART transmit pin.

TMS470 MCU project

Root TMS470 MCU project directory consists of following files and directories:

Name	**Type**	**Description**
hw_tms470.c	file	Implementing specific hardware functions (**_kbhit**, **_cputch**, **_cgetch** and **Init_MCU**)
cstartup.s	file	Contains low-level initialization of TMS470 MCU
Configuration_Files	dir	Contains configurations files (linker file, init files) for specific TMS470 MCU version.

Project was created using IAR Embedded Workbench IDE V5.4. Texas Instruments TMS470R1A128 MCU was used. More information about this MCU can be found at

http://focus.ti.com/docs/prod/folders/print/tms470r1a128.html.

Pin	Dir	Connect to uVGA	Description
C2SIaRX	OUT	RTS	Pin is set (written) high to stop MicroVGA sending data. Pin is set (written) low to allow MicroVGA sending data.
C2SIaTX	IN	CTS	Pin is read to find out, if MicroVGA is able to receive data. High means that MicroVGA is busy, low means ready to receive.
SCI1RX	IN	TXD	Standard UART receive pin.
SCI1TX	OUT	RXD	Standard UART transmit pin.

AVR (Atmel ATmega) MCUs project description

Root of AVR MCU project directory consist of following files and directories:

Name	Type	Description
hw_atmega128.c	file	Implementing hardware specific function (**_kbhit**, **_cputch**, **_cgetch** and **Init_MCU**)

Oscillator: internal 1 MHz