OVERVIEW
The following application note describes the interface of a LCD text display to a 8051 microcontroller system. This application note comes with the µVision2 project LCD_Display.UV2 that includes C source code of the interface software and a complete simulation script for simulator based testing. In this example, we are using the LCD-Controller ST6077. However, the interface to other text displays is similar.

µVision2 provides full integration of the C51 compiler and debugger. Today applications need to be tested before real hardware becomes available. Therefore µVision2 offers both: a simulator that seamless simulates the complete peripherals of countless 8051 derivatives, and a target debugger that allows software testing in real hardware with a target monitor or emulator.

If you open the µVision2 project LCD_Display.UV2 you can easily follow the steps described in this application note.

HARDWARE INTERFACE
This section gives an overview about the connection and communication between the ST6077 LCD display device and the 8051 microcontroller. Details are provided in the datasheet st6077v151.pdf that comes in the file APNT_161.ZIP.

The schematic below shows you how the devices are connected:

The data bus of the ST7066 is connected to the 8051 Port P0. The three control pins RS, RW, and E are connected to the 8051 I/O Port P2.0 – P2.1. These connections are reflected in the software interface.

<table>
<thead>
<tr>
<th>Overview of the display pins</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB0 – DB7</strong></td>
</tr>
<tr>
<td><strong>RW</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>RS</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
SOFTWARE INTERFACE

The file LCD_interface.C, that comes with the μVision2 project LCD_Display.UV2 is the source code that handles all basic input/output to the LCD text display. You may use this file as template for your own 8051 display driver.

```c
#include <reg51.h>
#include <stdio.h>
#include <intrins.h>
sbit P20_PIN = P2^0; // I/O Pin for P2.0
sbit P21_PIN = P2^1; // I/O Pin for P2.1
sbit P22_PIN = P2^2; // I/O Pin for P2.2
#define LINE1_ADDR 0x00 // Start of line 1 in the DD-Ram
#define LINE2_ADDR 0x40 // Start of line 2 in the DD-Ram
#define CLEAR_DISPLAY 0x01 // Clear entire display and set Display Data Address to 0
#define DD_RAM_PTR 0x80 // Address Display Data RAM pointer
#define DISP_INIT 0x38 // 8 bit 2 lines
#define INC_MODE 0x06 // Display Data RAM pointer incremented after write

static unsigned char ReadInstrReg (void) {
    unsigned char Instr;
    P0 = 0xff; // DATA PORT is input
    P20_PIN = 0; // select instruction register
    P21_PIN = 1; // read operation
    P22_PIN = 1; // give operation start signal
    _nop_(); _nop_(); // wait
    Instr = P0; // read Instruction
    P22_PIN = 0;
    return(Instr);
}

static void WriteInstrReg (unsigned char Instr) {
    P20_PIN = 0; // select instruction register
    P21_PIN = 0; // write operation
    P22_PIN = 1; // give operation start signal
    _nop_(); _nop_(); // wait
    P0 = Instr; // write instruction
    P22_PIN = 0;
    P0 = 0xff; // DATA_PORT is input [prevent I/O-Port from damage]
}
```

1: Data Register (read and write)

E: Enables data read/write (This has to follow each command)
static unsigned char ReadDataReg (void) {
    unsigned char val;
P0 = 0xff; // DATA_PORT is input
P20_PIN = 1; // select data register
P21_PIN = 1; // read-operation
P22_PIN = 1; // give operation start signal
    _nop_ (); _nop_ (); // wait
val = P0; // read Instruction
P22_PIN = 0;
return (val);}

/*
* WriteDataReg: Write to Data Register of LCD Display Device
*/
static void WriteDataReg (unsigned char val) {
P20_PIN = 1; // select data register
P21_PIN = 0; // write operation
P22_PIN = 1; // give operation start signal
    _nop_ (); _nop_ (); // wait
P0 = val; // write value
P22_PIN = 0;
P0 = 0xff; // DATA_PORT is input [prevent I/O-Port from damage]
}

char putchar (char c) {
    unsigned char line;
    if (c == '\n') {
        line = ReadInstrReg ();
        if (line & LINE2_ADDR) { // is already in line 2
            WriteInstrReg (LINE1_ADDR | DD_RAM_PTR); // place to start of line 1
        } else {
            WriteInstrReg (LINE2_ADDR | DD_RAM_PTR); // place to start of line 2
        }
    } else {
        WriteDataReg (c);
    }
    return (c);
}

//* Waits for some time so the 7066 can do the last command
*/
void wait (unsigned int time) {
    int i;
    for (i = 0; i < time ;i++){
    }
}

/*
* Clear_Display: Write 0x20 to Display RAM
*/
static void Clear_Display (void) {
    WriteInstrReg (CLEAR_DISPLAY);
    wait (1);
}

The putchar function is the basic output routine that is used for simple character output. These function is used by the function printf. In addition we have implemented the function Clear_Display.
The following test code allows you to check the implementation.

```c
void main (void) {
    unsigned char c;
    for (c = 'A'; c < 'H'; c++) { // Put the characters A to H to the display
        putchar (c);
    }
    printf ("Hello \n"); // Puts 'Hello' to the display using printf()
    Clear_Display (); //Clear the display and reset cursor
    for (c = 0; c < 10; c++) {
        printf ("This is Display Line %bd\n", c); // another printf() testing
    }
    while (1); //never stop
}
```

### Function Overview of the Software Interface

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ReadInstrReg()</code></td>
<td>Returns the instruction in the instruction register.</td>
</tr>
<tr>
<td><code>WriteInstrReg(Instr)</code></td>
<td>Writes the instruction <code>Instr</code> out to the instruction register. Nothing is returned.</td>
</tr>
<tr>
<td><code>ReadDataReg()</code></td>
<td>Returns the value in the Data Register.</td>
</tr>
<tr>
<td><code>WriteDataReg(val)</code></td>
<td>Writes the value <code>val</code> to the Data Register.</td>
</tr>
<tr>
<td><code>putchar(c)</code></td>
<td>Behaves like a normal putchar value. The value in <code>c</code> is written to the display and returned if it was successful.</td>
</tr>
<tr>
<td><code>Clear_Display()</code></td>
<td>Clears the entire Display by filling the Data Ram with blanks.</td>
</tr>
</tbody>
</table>

The test code does not use all interface functions, but they are implemented for completeness.

### µVISION2 SIMULATION

The following section shows you how to simulate the LCD text display with µVision2.

The µVision2 simulator provides several ways to simulate a text display. One of the easiest methods is to use the µVision2 memory window as ASCII display. In addition to the standard microcontroller memory, µVision2 offers four user defined memory spaces that are accessed with the prefixes `S:`, `T:`, `U:`, and `V:`. We are using the `U:` memory space as display memory for the LCD text display. The µVision2 memory window is used in ASCII mode. You may resize the window, so that it reflects the size of your LCD display as shown below.

The following µVision2 script defines debug functions and other required settings to simulate the LCD text display.
The MAP command reserves 80 Bytes of the user-memory U:. The size is identical to your display size. In this example:

\[
2 \text{ rows} \times 0x40 \text{ characters} = 0x80 \text{ character with 1 Byte each}
\]

The debug function `Simulate_Display` is triggered on a write to Port 2 (P2). This behavior is defined with the `BreakSet` command on the last line of the µVision2 script. This command connects the control lines of the LCD text display controller to the simulated 8051 device. Any write event on Port 2 (P2) calls the function `Simulate_Display`. Depending on the previous Port 2 (SD_oldPORT2) state, a LCD display read or write command is performed as explained in the following diagram.

- Read from LCD display: E (P2.2) signal changes from low to high.
- Write to LCD display: E (P2.2) signal changes from high to low.

The signal RS (Port 2.0) selects the instruction register or the display RAM of the LCD device. The signal RW (Port 2.1) defines a read or write command to the LCD device.
if ((SD_oldPORT2 & 0x04) && !(PORT2 & 0x04)) { // LCD write: E changes to low
    if ((PORT2 & 0x03) == 0) { // write to command register
        if (PORT0 == 0x01) { // command is 'Clear Display'
            SD_clear (); // call the 'Clear Display' simulation
        }
    }
    if (PORT0 & 0x80) { // Set DDRAM address (cursor position)
        SD_cursor = (PORT0 & 0x7F); // set DD RAM pointer
    }
}
if ((PORT2 & 0x03) == 0x01) { // command is 'Write to RAM'
    _WBYTE (U:0+SD_cursor, PORT0); // write to DD RAM (user memory @ U:0xXX)
    SD_cursor++; // Advance cursor position by 1
}
if (PORT2 & 0x02) { // read mode
    PORT0 = 0xFF; // display data goes to high
}
SD_oldPORT2 = PORT2; // store E for change detection

BS WRITE PORT2, 1, "Simulate_Display ()"

The debug function Simulate_Display analyses some of the states of the LCD display device. To simplify the debug function, we have only implemented the commands that are used by the C51 software interface. The variable SD_cursor holds the cursor position in the DDRAM (display memory) and is used as pointer to the user memory U:

CONCLUSION
Implementing an LCD text interface is straightforward with the Keil development tools. It is even possible to simulate the whole target system with µVision2 without sacrificing execution speed. The simulation model described in this application note may be adapted to most other LCD text display controllers.

This application note uses the µVision2 debug functions as peripheral interface. Application Note 154 “Implementing µVision2 DLL’s for Simulating User Defined Hardware” describes an interface that is able to simulate complex peripherals which require accurate timing and/or several input/output variables.