UVI51 USER'S MANUAL (v 5.0)

Alfredo del Río
Dpto. de Tecnología Electrónica
ETSII-Universidad de Vigo
36200 VIGO
Spain (Europe)

Note: A Spanish version of UVI51, software and manual, is available in the site http://www.dte.uvigo.es

Contents

1. Introduction

UVI51 is the 8051 microcontroller teaching/learning environment developed in the University of Vigo (Spain), *Dpto. de Tecnología Electrónica*.

The present manual is delivered with the free distribution version (UVI51 v5.0). This version limits the size of source files to 500 lines. The commercial version puts this limit to 3000 lines.

This environment includes a set of programs and auxiliary files. The main program is the UVI51 assistant. The assistant leads the user through the software development and verification processes. The assembler UVIENS51 generates the executable code and the simulator UVISIM51 helps in program verification. This programs are specially designed to work together. However, the user can configure the system to use other commercial assemblers (see appendix A2).

The next sections show a brief description of the various elements included in UVI51.

The present manual is not intended as a guide to 8051 microcontrollers. Furthermore, a basic knowledge on this family of microcontrollers is needed to follow some parts of the manual. UVI51 includes a set of source files that can be used as examples, or even as a minimal course on 8051.

2. UVI51 operation modes

UVI51 can work in two different modes called *single-micro* and *multi-micro*.

The *single-micro* mode is faster and simpler to use. However only one microcontroller can be simulated in this mode. Moreover, the user can not define external peripherals.

The *multi-micro* mode is much more powerful. The user can define a system composed of up to four micros plus a set of external peripherals as pushbuttons, switches, LEDs, displays and a logic analyzer. System definition is accomplished through a *system configuration file* with the CNX extension.

In *multi-micro* mode the following parameters can be defined for each micro in the system:

- Clock frequency.
- Program memory size (ROM).
- External RAM memory size (XRAM).

Appendix A1 includes a detailed description of configuration files syntax.

3. Installing UVI51. What is new in version 5.0?

This software is supplied as a packed file named UVI51_50.ZIP. This ZIP file must be unpacked previously to installation procedure.

This UVI51 version is a free distribution software. However, it must be distributed as a whole and no part must be modified.

All the programs included in UVI51 are MSDOS applications and do not require any special installation. The set of files resulting of the unpacking process must be copied into a subdirectory. The creation of a work subdirectory for work files (system definition files, source files, etc) is suggested, but not a must.

Installation and execution under Microsoft Windows’95 and later versions

The best way to start UVI51 under Windows is creating a shortcut to UVI51.EXE (the UVI51 assistant).

The invocation line must include a full path to the UVI51 directory followed by UVI51.EXE. For the use of parameters see section 4.

The suggested shortcut properties are:

PROGRAM- Close at exit; Change icon- UVI51.ICO

SCREEN- Full screen

MISCELLANEOUS- Do not use screen saver. Low inactive sensitivity

New in version 5.0

- New simulator graphical interface including windows and buttons.
- The simulator enables now the continuous execution while showing and updating the source code or the embedded peripherals windows. These windows let the user to edit values.
- The simulator ‘RAM’ window has been enhanced to add a new display format as a symbol table.
- The external peripheral display present in older versions has been changed to a much more elaborated and graphical display. The new display indicates what nodes each peripheral is connected to.
- A new external peripheral named *pushbutton* has been added. Moreover, the old syntax of CNX files has been extended to include peripheral labels. **Appendix A1 describes this syntax**.
- New assembler version with a more powerful set of directives. In addition, this version eliminates some errors found in previous ones.
- The simulator source window can be synchronized or desynchronized to the execution process.
- New *Find* utility for searching text in the source window.
- Simulation speed is now adjustable.

4. The Assistant

The assistant is UVI51's central core. The user starts UVI51 calling the assistant from DOS prompt or using a Windows shortcut.

One of the following command lines can be used to run the assistant from DOS prompt.

```
UVI51

UVI51 file.cnx

UVI51 file.src
```

The first two lines configure UVI51 to work in multi-micro mode (see section 2). The first line starts the assistant with the NONAME.CNX system configuration file, while the second employs the indicated file (file.cnx in this case).

The third line configures UVI51 in single-micro mode. This mode does not use any configuration file. The indicated file name is considered as the source file for the only micro to be simulated. The extension must be SRC. The micro is configured to work at 12 MHz clock frequency (1 μs machine cycle) with a 4096 bytes ROM and a 2048 bytes XRAM.

Once the assistant has started in one mode, it can not be changed to the other. The only way to change the working mode is to start the assistant again with the suitable DOS command line (or Windows shortcut).

If the file name passed in the invocation line includes a path, that path is established as the user files default path.

4.1. Multi-micro mode

When working in multi-micro mode, the assistant screen shows the following buttons:

[HELP] Opens the help file HELP.TXT. This file is supplied containing information about the 8051. The user can edit the file according to his or her needs.

[OPEN] Opens an already existing configuration file (CNX). This file will lead the assistant work.

[NEW] Opens a new configuration file (CNX). The user must edit the file to define the system before the system can be simulated (see the next button).

[file.CNX] This button is designated with the name of the open CNX file. Push it to edit the file.

[SIMULATE] This button is active only if the CNX file is valid and all the source files have been succesfully assembled. This button calls the simulator.

[ANALYZE] This button calls the logic analyzer viewer. It is active only if a logic analyzer has been defined in the configuration file and a simulation has been successfully carried out previously (see appendix A1). The viewer shows the chronograms of all the nodes in the system.

[EXIT] This button quits the assistant and returns to the operating system.

In some versions other buttons are included (for example [DEBUG] or [TERMINAL]).

In addition, if the CNX file is right, a table of buttons is shown with the format:

SOURCES	ASSEMBLER	OUTPUT
[file1.SRC]	[--A->]	[file1.LST]
[file2.SRC]	[--A->]	[file2.LST]

The buttons in the left column are used to edit the source files in assembly language. The buttons in the central column call the assembler on the corresponding source file. Finally, the buttons in the right column are used to edit the list files generated by the assembler.

Once a source file is modified, and while it is not successfully assembled, the associated LST file is shown in gray. Even so, the LST file can be edited using the button in order to see, for example, the errors of the last assembly process.

The following line is shown in the bottom of the screen:

STATUS MESSAGES:

The assembler, or other utilities, show their console messages under this line.

The same area is also utilized when creating a new CNX file using the [NEW] button. The name of the new file is written there.

4.2. Single-micro mode

When working in single-micro mode, the assistant shows a screen similar to that of the multi-micro mode (see section 4.1). A brief description of the differences follows:

[OPEN] Open a source file (SRC).

[NEW] Create a new source file (SRC).

[SIMULATE] If the source file has been successfully assembled, this button calls the simulator.

The table of buttons contains in this case a single line (only one micro), and is functionally equivalent to that of the multimicro case.

SOURCE	ASSEMBLER	OUTPUT
[file1.SRC]	[--A->]	[file1.LST]

5. The Assembler

5.1. Introduction

UVIENS51 is a ‘mini-assembler’ for the 8051 family of microcontrollers. It has been developed as a part of UVI51. This assembler is very easy to use and employs a syntax very similar to that of another commercial assemblers like Intel’s ASM51. UVIENS51 generates absolute code directly, and there is no need of a separate linker. The definition and use of macros is not allowed in this version.

5.2. Running the assembler

From the assistant, click the [--E->] button to run the assembler. UVIENS51 is an MSDOS application, therefore it can be also called from the MSDOS command line:

```
UVIENS51 file.src
```

being *file.src* the name of the source code file to be assembled. A path can be included in the file name.

The assembler creates an output file with the same name as the source file but LST extension. If no error is found, another file is created with the HEX extension containing the absolute code in Intel-Hex format.

The LST file is useful to inspect the errors the assembler can eventually find. Moreover, the simulator needs this file to correlate source to object code.

5.3. Assembly language syntax

The source file must be an ASCII file. Since UVIENS51 generates absolute code directly, it does not accept some directives commonly used by another assemblers. A list of valid directives and formats follows:

DIRECTIVE	**FUNCTION**
ORG expression	Updates *program counter* to the value of *expression*
symbol EQU expression	Defines a symbol of type *number* with the value of *expression*
symbol BIT expression	Defines a symbol of type *bit address* with the value of *expression*
symbol DATA expression	Defines a symbol of type *direct byte address* (direct internal RAM) with the value of *expression*
symbol IDATA expression	Defines a symbol of type *indirect byte address* (indirect internal RAM) with the value of *expression*
symbol XDATA expression	Defines a symbol of type *external byte address* (external RAM) with the value of *expression*
symbol CODE expression	Defines a symbol of type *code address* (ROM) with the value of *expression*
DB expression	Generates a byte of code with the value of *expression*
DB 'c'	Generates a byte of code with the value of the indicated character ascii code.
DB "string"	Generates a byte of code for every character in the string with the value of their ascii codes. The backslash \ must precede single or double quotes to be included inside the string.
DB ‘a’, 25 , "string"	Generates bytes of code for all the elements in the list.
DW expression	Generates 2 bytes of code with the value of *expression*, the upper byte first.
END	Indicates the end of file (not required)
#IF expression	If *expression* evaluates to 0, the following lines are not assembled until the directive #ENDIF is found (conditional assembly).
#ENDIF	Restores the unconditional assembly.

All the expressions must be absolute and computable at the first pass (only backward references are permitted). The only exception are the forward reference to labels when using EQU-style directives and, of course, in *jump* and *call* instructions.

The syntax is similar to that used by other assemblers. Some examples follow:

```
label:

<label:>  mnemonic  parameter_list   ;optional remark

<label:>  mnemonic  ;optional remark

<label:>  ;optional remark

;remark line without instruction
```

Valid mnemonics are those defined by Intel for 8051 family. If two or more parameters are used in one instruction, they must be separated by commas. If a instruction accepts a numeric parameter, a computable expression can be used. All expressions are evaluated into numbers in the range 0000h to 0FFFFh. The evaluation is performed left to right without precedences, so parenthesis must be used to define the desired precedence. The following operators are accepted:

+	Add
-	Subtract and also the *change sign* unary operator
*	Unsigned multiplication
/	Quotient of whole division
%	Remainder of whole division
&	Logic AND
|	Logic OR
~	Logic NOT, equivalent to 1's complement
^	Exclusive OR
<	Shift to left one bit position
>	Shift to right one bit position
[	Most significant byte of operand (similar to Intel's HIGH)
]	Least significant byte of operand (similar to Intel's LOW)

The evaluation is performed using 16-bit values. Negative values are represented in 2's complement notation. The result of an operation is assigned the type that follows:

- The same type as the operand in unary operators.
- When *adding* or *subtracting*, if one operator is a generic number and the other is not, the result is of the same type as the non numeric operator.
- Generic number –as if it were defined using the EQU directive- in other case.

When using jump or call instructions, the absolute destination address must be indicated –usually in the form of a label-, never a relative distance. The assembler computes the relative jump if necessary.

The special symbol $ can be used to refer to the current value of program counter. Therefore, SJMP $ indicates a jump to itself. If a DB directive generates several bytes, the symbol $ stands for the address where the first byte will be located.

The generic JMP and CALL mnemonics are recognized by the assembler. The shortest valid jump is selected by the assembler in backward references. In forward references, the assembler selects always the long alternative.

The assembler converts the source file into uppercase. There are only two exceptions:

- Comments
- Characters between single or double quotes

5.4. Assembler restrictions

The following limits have to be observed when using this assembler version:

- The maximum number of symbols is 1000. These include the symbols defined in the UVI51.REG file.
- Source file size is limited by the amount of free conventional memory available for DOS applications. As a guideline, the maximum number of lines is about 10000.
- Lines of source code are truncated to 80 characters.
- Symbols and labels are truncated to 31 characters.
- Parameter lists are truncated to 31 characters in both instructions and directives. This should be in mind especially when using long DB directives.

6. The Simulator

This section discusses simulator features and usage.

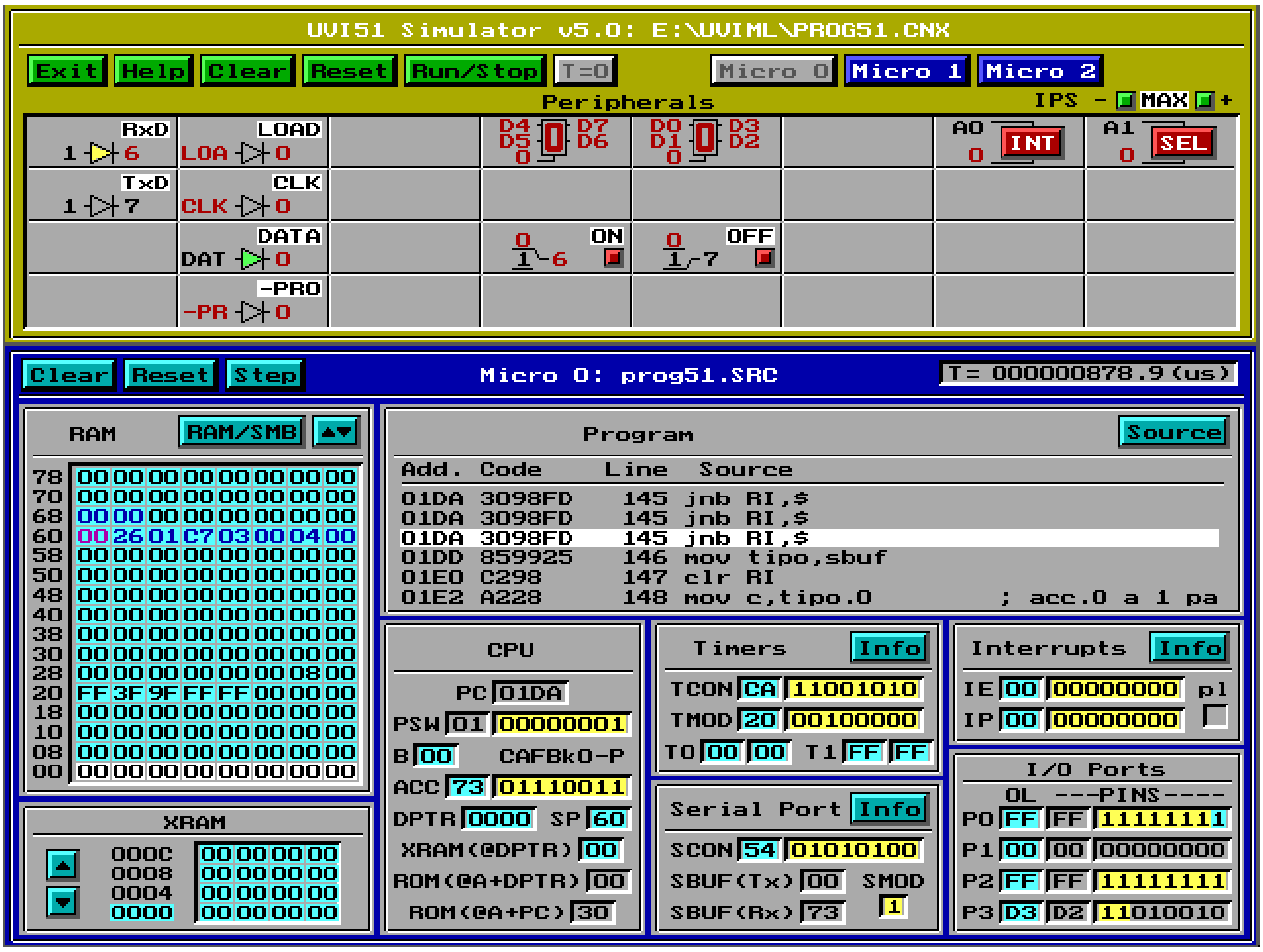
6.1. Main window and basic topics

The main features of UVISIM51 v5.0 are:

- Multimicro systems simulation.
- Graphic user interface.
- *Step over*, *trace into* and *continuous run* options.
- Unlimited breakpoints.

Both mouse and keyboard can be used to control the simulator. Some options require the use of mouse.

The simulator main window is shown in the following figure.



```
Add.  Code      Line  Source
01DA  3098FD     145  jnb RI,$
01DA  3098FD     145  jnb RI,$
01DA  3098FD     145  jnb RI,$
01DD  859925     146  mov tipo,sbuf
01E0  C298       147  clr RI
01E2  A228       148  mov c,tipo.0                ; acc.0 a 1 pa
```

```
RAM
78 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00
68 00 00 00 00 00 00 00 00
60 00 26 01 C7 03 00 04 00
58 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00
48 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00
38 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00
28 00 00 00 00 00 00 08 00
20 FF 3F 9F FF FF 00 00 00
18 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00
08 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00

XRAM
000C 00 00 00 00
0008 00 00 00 00
0004 00 00 00 00
0000 00 00 00 00
```

```
CPU
         PC 01DA
PSW 01 00000001
B 00       CAFBkO-P
ACC 73 01110011
DPTR 0000   SP 60
XRAM(@DPTR) 00
ROM(@A+DPTR) 00
 ROM(@A+PC) 30

Timers
TCON CA 11001010
TMOD 20 00100000
T0 00 00   T1 FF FF

Serial Port
SCON 54 01010100
SBUF(Tx) 00   SMOD
SBUF(Rx) 73      1

Interrupts
IE 00 00000000   pl
IP 00 00000000

I/O Ports
    OL  ---PINS----
P0 FF FF 11111111
P1 00 00 00000000
P2 FF FF 11111111
P3 D3 D2 11010010
```

The upper window, in orange –or brown- is called *system window*. Its title shows the name of the system being simulated. Under the title there is a set of buttons related to system operation. Under the buttons, the external peripherals are drawn in a set of cells (see appendix A1). A brief description of the system window buttons follows:

Exit	Quit the simulator. Equivalent to Shift+Esc.
Help	Show a brief help. Equivalent to F1.
Clear	Reset all the micros in the system, clear their RAMs, reset the global time clock and clear all the breakpoints. Equivalent to 'C' (uppercase).
Reset	Reset all the micros in the system. Equivalent to 'R' (uppercase).
Run/Stop	Run the simulator or stop it. Equivalent to 's' (lowercase).

Any of the following conditions also stops the simulator:

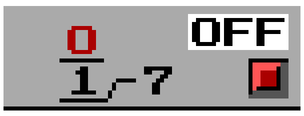
- The program counter is pointing to a non initialized ROM address. In this case, a real micro should go into an undetermined state.
- A breakpoint is reached.
- The reserved A5h code has been found.

T=0 Resets the global time clock. This button is inhibited if a logic analyzer has been defined in the CNX file.

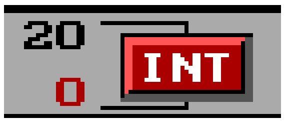
Micro 0 Micro 1 Select which micro is to be displayed in the *microcontroller window* (see below). There are as many buttons as micros have been defined in the system, never more than 4. As an exception, when only one micro has been defined, no button of this kind is shown.

The external peripheral area shows the state of every peripheral defined in the CNX file. (see appendix A1). Each peripheral type has its own characteristic picture. Node numbers are shown either in black -to indicate a high level state- or in red –low level state-. Numbers are substituted by labels –node names- if so defined in the CNX file.

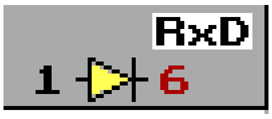
- Manually operated switch. The switch name and electrical symbol are shown. The moveable contact is shown in its current position. Switch state is changed using the little button. **Node controlled switches are not shown in the peripherals area.**



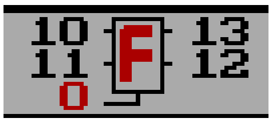
- **Pushbutton (key). This peripheral is new to the version 5.0**. A pushbutton is shown with the assigned name. The contacts are closed while the button is pushed. Since only one mouse pointer is available, only one pushbutton can be pushed at a time. Would this be a problem, replace pushbuttons with switches.



- Logic test -LED diode-. This peripheral is shown using the LED symbol. When active, it is shown lighting. The color is red by default but can be defined for each LED in the CNX file.



- Hex display -7 segments style- with multiplexing capability. Active segments are shown in red. When the -enable node is at '1' no segment lights.



The CNX file defines the place where every external peripheral is to be shown.

The system window includes a *simulation speed* indicator. The units are Instructions Per Second. The speed can be modified using the '+' and '-' buttons. When maximum speed is selected, the indicator shows MAX. The numeric value is shown in red when the requested speed is greater than the maximum possible for the computer.

The lower window, in blue, is called *microcontroller window*. There is a set of subwindows inside, each showing information on some different element of the selected micro.

The micro window header indicates the number -0 to 3- of the micro in the system and the name of its source file. Furthermore, the header includes 3 buttons and an elapsed time indicator. These buttons are:

Clear Resets the selected micro and clears its RAM. Equivalent to 'c' (lowercase).

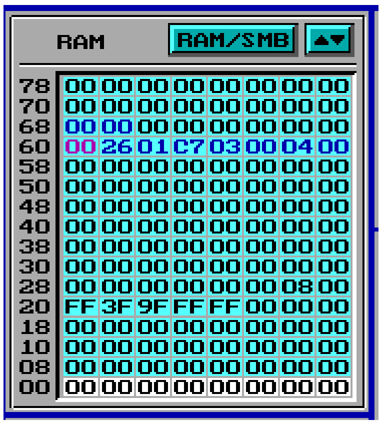
Reset Resets the selected micro. Equivalent to 'r' (lowercase).

Step Executes one instruction *tracing into* subroutines. If pushed while shift key is depressed, it executes one instruction *stepping over* subroutines.

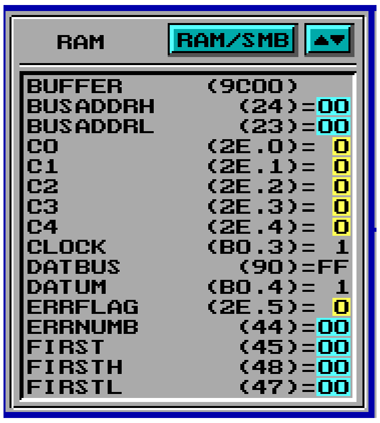
There is a set of subwindows under the header.

- The RAM window shows the contents of the internal RAM as a table. The *arrows* button switches between the 128 lower and 128 upper –indirect- bytes. It is worth noting that the upper 128 bytes do not exist in the basic 8051 micro. Values are shown in hexadecimal.

The RAM/SMB button switches between the just described tabular format and a list of symbols. The *PgUp* and *PgDn* keys can be used to scroll the list.

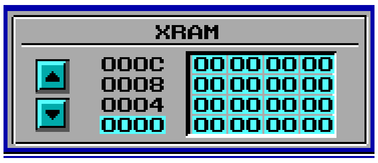


```
RAM            RAM/SMB  ▲▼
78 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00
68 00 00 00 00 00 00 00 00
60 00 26 01 C7 03 00 04 00
58 00 00 00 00 00 00 00 00
50 00 00 00 00 00 00 00 00
48 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00
38 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00
28 00 00 00 00 00 00 08 00
20 FF 3F 9F FF FF 00 00 00
18 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00
08 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00
```



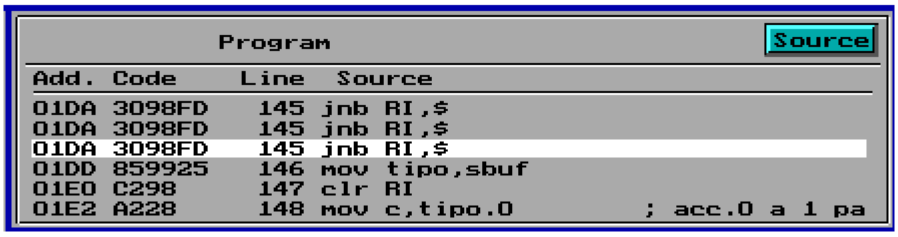
```
RAM            RAM/SMB  ▲▼
BUFFER          (9C00)
BUSADDRH          (24)=00
BUSADDRL          (23)=00
C0              (2E.0)= 0
C1              (2E.1)= 0
C2              (2E.2)= 0
C3              (2E.3)= 0
C4              (2E.4)= 0
CLOCK           (B0.3)= 1
DATBUS            (90)=FF
DATUM           (B0.4)= 1
ERRFLAG         (2E.5)= 0
ERRNUMB           (44)=00
FIRST             (45)=00
FIRSTH            (48)=00
FIRSTL            (47)=00
```

- The XRAM window shows the contents of 16 addresses of the external RAM. The arrow buttons are used to scroll the table. In order to reach quickly any XRAM address, the displayed pointer –over blue background- can be edited.



```
        XRAM
▲  000C  00 00 00 00
   0008  00 00 00 00
▼  0004  00 00 00 00
   0000  00 00 00 00
```

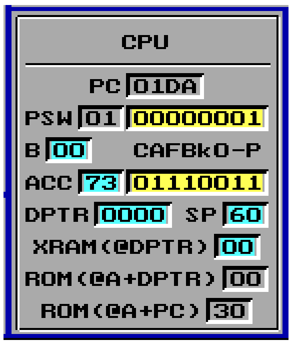
- The Program window shows the contents of program memory. Code addresses are shown in the first column while code itself is shown in the second. The respective source file lines are shown preceded by a line number.



```
              Program                            Source
Add.  Code      Line  Source
01DA  3098FD     145  jnb RI,$
01DA  3098FD     145  jnb RI,$
01DA  3098FD     145  jnb RI,$
01DD  859925     146  mov tipo,sbuf
01E0  C298       147  clr RI
01E2  A228       148  mov c,tipo.0            ; acc.0 a 1 pa
```

Source This button opens a new window, called Source window, which shows the source code in more detail, including the lines that do not generate code, as remarks or some directives. Equivalent to the key 'F' (uppercase). Section 6.2 describes source window.

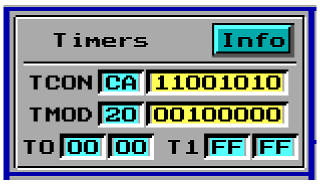
- CPU window shows the contents of some registers associated to the *central process unit*. The contents of some interesting XRAM and ROM addresses are also shown. Some registers are shown both in hexadecimal and binary.



```
        CPU
      PC 01DA
PSW 01 00000001
B 00     CAFBkO-P
ACC 73 01110011
DPTR 0000  SP 60
  XRAM(@DPTR) 00
ROM(@A+DPTR) 00
   ROM(@A+PC) 30
```

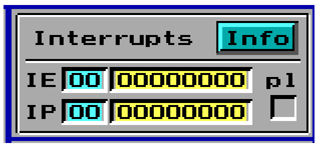
- The Timer/Counter window shows the contents of TCON, TMOD and the pairs TH0-TL0 and TH1-TL1.

Info This button opens a window which shows the state of timers in a graphic and more detailed way. For more information, see section 6.3.



```
 Timers          Info
TCON CA 11001010
TMOD 20 00100000
T0 00 00  T1 FF FF
```

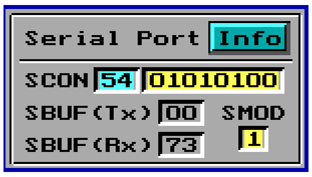
- The Interrupts window shows the contents of IE and IP. Moreover, an indication of priority level of the interrupt being serviced is provided. The notation used for pl is L for low priority, H for high priority and blank when no interrupt is being serviced.



```
Interrupts       Info
IE 00 00000000   pl
IP 00 00000000
```

Info This button opens a new window which depicts the state of the interrupt controller in a graphic and detailed way. See section 6.4.

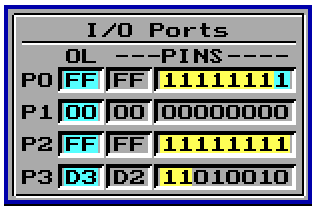
- The Serial Port window shows the contents of SCON, SBUF(tx) and SBUF(rx). Even though only one address is associated to the symbol SBUF, two SBUF registers exist in the serial controller. One of them is located in the transmitter section and the other one in the receiver section. The value of the bit SMOD – PCON.7- is also shown.



```
Serial Port [Info]
SCON 54 01010100
SBUF(Tx) 00   SMOD
SBUF(Rx) 73    1
```

Info This button opens a new window where the state of the serial controller is shown. See section 6.5.

- The I/O Ports window shows the state of port latches and terminals. The latter are shown in both hexadecimal and binary format. The latch value might not match the terminal levels, due to the use of quasi bi-directional ports, as indicated in the 8051 manuals. The externally applied level can be modified for each terminal using either the mouse or the keyboard. With the mouse, the applied level toggles clicking the desired bit. With the keyboard, the blue cursor can be moved using the arrow keys, and then the space bar depressed to toggle. **CNX files usually assign node numbers to some port terminals. The state of that terminals can not be modified in the way just described. These terminals are shown on a gray background. However, they can be modified using pushbuttons or switches (external peripherals).**

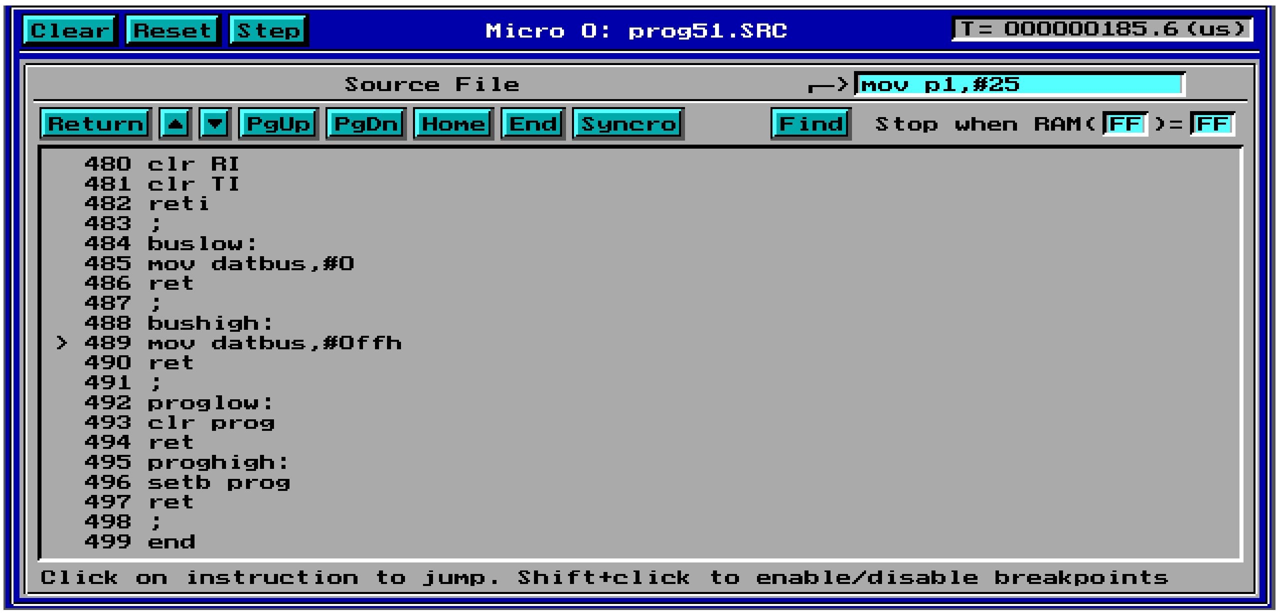


```
     I/O Ports
   OL  ---PINS----
P0 FF FF 11111111
P1 00 00 00000000
P2 FF FF 11111111
P3 D3 D2 11010010
```

Inside most of windows, many registers can be edited using the mouse, at byte and/or bit level. The same is valid for RAM and XRAM contents. Byte editable values are shown on cyan background. Bit editable values are shown on yellow background. See sections 6.6 and 6.7 for more details.

6.2. Source file window.

The button [Source], in the Program window, opens the Source window. This window hides the set of subwindows discussed in the preceding sections.



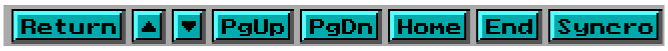
```
Clear Reset Step          Micro 0: prog51.SRC           T= 000000185.6(us)

                 Source File                  ─> mov p1,#25
Return ▲ ▼ PgUp PgDn Home End Syncro       Find   Stop when RAM(FF)=FF

   clr RI
   clr TI
   reti
   ;
   buslow:
   mov datbus,#0
   ret
   ;
   bushigh:
 > mov datbus,#0ffh
   ret
   ;
   proglow:
   clr prog
   ret
   proghigh:
   setb prog
   ret
   ;
   end
Click on instruction to jump. Shift+click to enable/disable breakpoints
```

The Source window shows the source code in assembly language, including instructions, directives, remarks, blank lines, etc.. The next instruction to be executed is marked with the cursor ‘>’.

The following set of buttons is placed below the header.



Return — Returns to the default microcontroller window –the set of subwindows-. Equivalent to ‘Esc’ key.

▲ ▼ — Scrolls the list up and down. Equivalent to the up and down arrow keys.

PgUp PgDn — Scrolls a whole page up and down respectively. Equivalent to the equally named keys.

Home End — Moves to the first or last line of the program. Equivalent to the keys ‘Home’ and ‘End’.

The latter 6 buttons are utilized to display the desired program portion, leaving the program desynchronized. This means the window will not follow program execution and therefore the cursor ‘>’ will probably disappear.

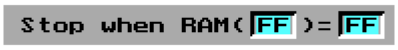
Syncro — This button can be used to resynchronize the window to program execution.

The following operations can be performed within this window:

- Jump to instruction. Clicking a program line –which generates code, usually an instruction- the program counter is updated with the address of that instruction. Therefore, the selected instruction will be the next to be executed.

- Toggle a breakpoint. If the same operation is performed but keeping the shift key depressed, a breakpoint is established in –or removed from- that instruction. Breakpoints are shown on a red background.

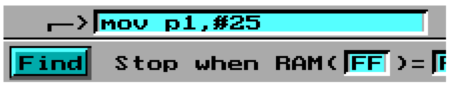
- Set a *value breakpoint*. Two editable cells are placed in the right side of the header, as shown:



  Program execution will stop when the stated condition becomes true. The value between brackets indicates a RAM address –if less than 80hex- or SFR address –if greater than 7Fhex-. This process is named a *value breakpoint*. Only one can be defined at a time.

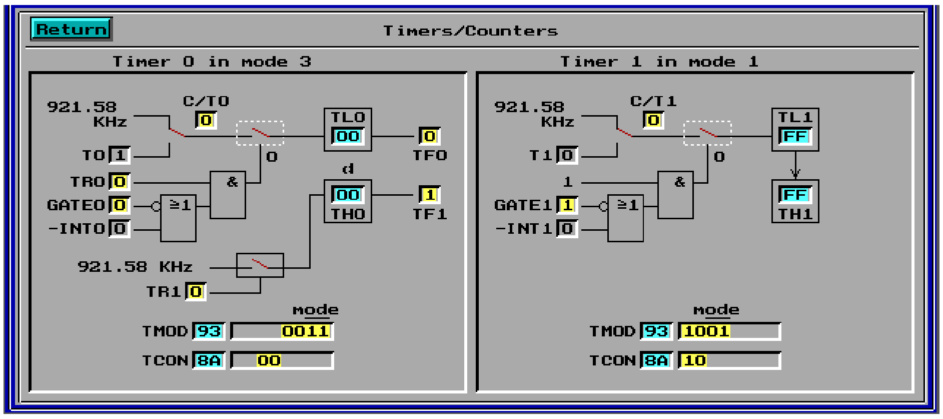
  The value breakpoint mechanism can be disabled by simply selecting a non existing address with a non reachable value. For example RAM(FF)=FF, since the FFhex address indicates a non existing SFR with a default value of 0.

There is a text search utility in the source code window. Click inside the text box to edit the search pattern. Click the [Find] button to start search. The search begins at the first line of the current window. If a match occurs, the window scrolls to show it placed at the first line. If the pattern is not found, no action takes place. Since the window never scrolls down below the end of the program, patterns found at the end of the program might not show the match at the first line.



6.3. Timers window

This window shows a functional diagram of timers including current configuration and state. A line of text indicates the operation mode. Timer control bits are placed where they act, and also grouped into TMOD and TCON registers. Their values can be edited, and their effect over timer configuration is updated immediately. This window is also continuously updated when the simulator is in running mode. In this mode, however, editing values is not permitted.


Timers/Counters

Timer 0 in mode 3

Timer 1 in mode 1

TMOD 93 0011

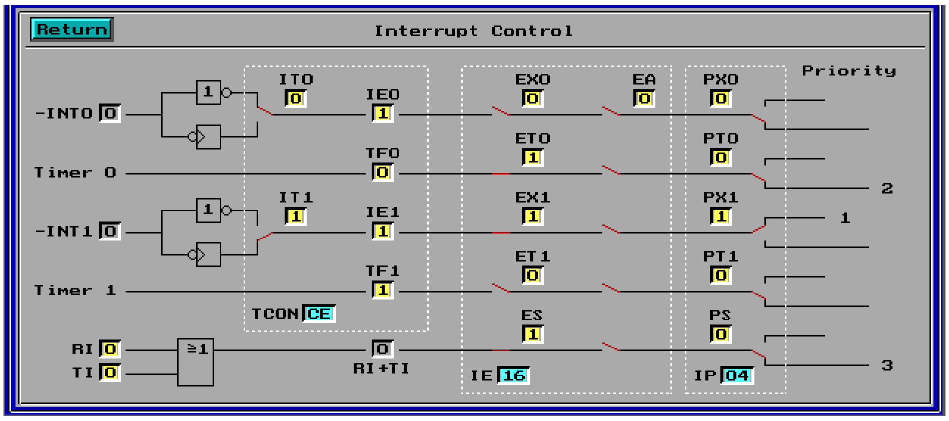
TCON 8A 00

TMOD 93 1001

TCON 8A 10

6.4. Interrupts window.

This window shows a functional diagram of the interrupt controller indicating current configuration and state. The diagram shows the contents of the IE and IP registers both as a whole and bit by bit. Edition of these registers is enabled except in running mode. With a sight, the user can determine what interrupt sources are enabled and which of them are to be serviced first. Window contents are updated continuously when the simulator is in running mode.


Interrupt Control

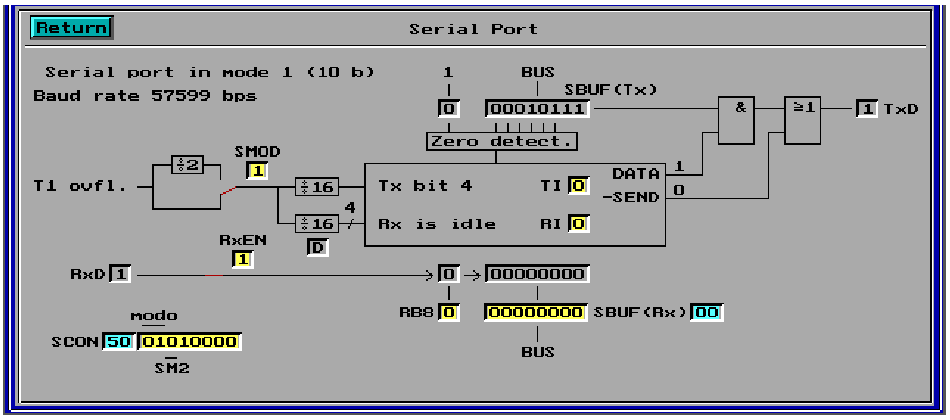
TCON CE

IE 16

IP 04

6.5. Serial Port window.

This window shows a functional diagram of the serial controller. Modes 0 and 2 are not supported by the current version of UVI51. The diagram indicates the mode, number of bits and, if possible, the baud rate. The state of transmitting and receiving processes are also shown.


Serial Port

Serial port in mode 1 (10 b)
Baud rate 57599 bps

Tx bit 4   TI 0
Rx is idle RI 0

SCON 50 01010000

In the transmitter portion, the currently being transmitted bit is indicated.

In the receiver portion, the last received –sampled- bit is shown. The state of the internal divider –0 to F- is also shown. RxD line is sampled by the 7, 8 and 9 states, as it is in an actual 8051.

The state of SBUF(tx), SBUF(rx) and the internal receiving shift register is also depicted. The SBUF(tx) acts itself as a shift register for the transmitter portion. SBUF(rx) is loaded with the 8 least significant bits of the shift register at the end of the frame, as in an actual 8051.

This window is updated periodically when the simulator is in running mode. In this mode, the edition of values is not permitted.

6.6. Use of colors

Colors have been assigned in simulator windows using the following guidelines:

GRAY: Non editable values background. Inactive buttons. LEDs in off state.

WHITE: Background color for the selected register bank. Background for the next instruction in Program window. Background for labels in external peripherals.

CYAN: Background color for byte editable values. Background for the cursor in the I/O Ports window. Microcontroller window buttons.

RED: Byte being edited. Background for breakpoints. Buttons in external peripherals. Hexadecimal display segments.

YELLOW: Background for bit editable values.

MAGENT: RAM address pointed by the stack pointer –SP-.

BLUE: Stack footprints –RAM area written by PUSH, ACALL or LCALL instructions, useful for stack overflow detection-. Microcontroller selection buttons.

GREEN: System window buttons –except for microcontroller selection buttons-.

6.7. Editing values

When in running mode, only the I/O ports state –external pin levels- can be changed manually. This operation can be performed using the keyboard or the mouse. With the keyboard, use the arrow keys to move the I/O Ports cursor shown in cyan color, then the space bar to toggle pin state. With the mouse, click on one pin to toggle.

When in step mode, RAM, XRAM and many registers can be edited. Some of them can be edited at byte level –byte editable- using hex notation. Click on the desired byte –the value is now shown in red color- and use the keyboard to enter the new hex value. End with the *Enter* key to accept changes, *Esc* to discard. Alternatively you can click outside the value using the left button to accept, right button to discard.

As a rule, the editable bytes are shown in cyan background color. All the RAM is editable, including the registers of the selected bank, though they are shown in white background color.

The following registers are byte-editable:

- ACC, B, DPTR -high and low bytes- SP, IE, IP, TCON, TMOD, T0 and T1.
- P0, P1, P2 y P3 (output latch value only).

Other byte-editable contents are:

- XRAM pointed by DPTR.
- XRAM display pointer (2 bytes) and the contents of XRAM shown in the corresponding window.

Some registers are bit-editable. Their binary values are shown in yellow background color. Click the bit with the mouse to toggle. The following registers are bit-editable:

- ACC, PSW (unless parity bit), IE, IP, SCON, TCON and TMOD. The bit SMOD too.
- P0, P1, P2 y P3 (external value applied to unconnected pins).

6.8. What 8051 configuration is simulated?

An 8051 with 4 Kbytes of ROM and 2 Kbytes of XRAM is simulated by default. This XRAM is connected using only the 11 least significant address lines. As a result, the external memory map is cyclical being any address *ADDR* over 2 Kbytes redirected to *ADDR* & 7FFh.

It is worth noting that a real application would connect the external ram via P0 and P2 I/O ports, used as buses. In such an application, P0 loses its early state and must not be used for port purposes. P2 changes only momentarily. However, the simulator neither show the bus activity in ports P0 and P2, nor leave P0 modified. The user must be aware of the restrictions in ports utilization when accessing external data or program memory.

On the other hand, the simulator takes into account P2 state when executing the following instructions: movx a,@r0 movx a,@r1 movx @r0,a movx @r1,a.

It is also worth noting that the simulator includes 128 bytes of indirect addressable internal RAM, not available in the original 8051 but in the 8052 and other derivatives. However, other 8052 improvements, as timer 2, are not implemented.

The current version simulates serial port in modes 1 and 3 only. The simulation is performed at the logic level of lines P3.0 and P3.1. Thus, a serial communication between 2 or more micros can be simulated. Furthermore, software UARTs can be mixed with the 8051 embedded serial port when simulating serial communications. Simulation realism includes the triple sample mechanism of the serial receiver.

7. The Logic Analyzer

7.1. Introduction

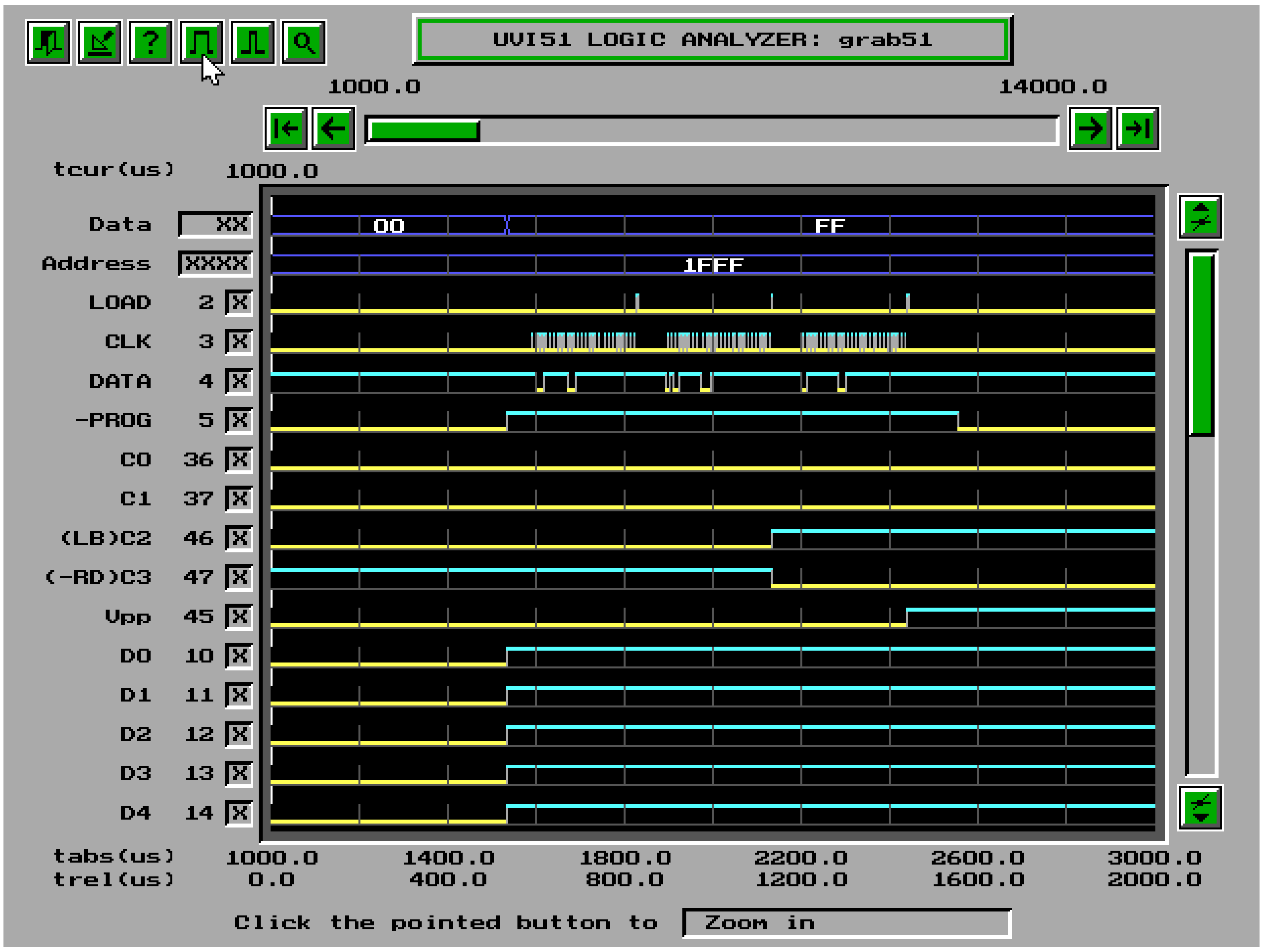
UVI51 includes the possibility of logging chronograms for every node in the system, while simulation is performed. The user must define a peripheral of type *logic analyzer* in the CNX file in order to use this feature (see appendix A1, section A1.2).

Once a simulation is performed, the stored chronograms can be displayed clicking the [ANALYZE] button which invokes the program GRAF51.EXE called *UVI51 logic analyzer*. This option is not available in single micro mode.

7.2. Description

The logic analyzer display is an MSDOS application. It can be handled using the mouse or the keyboard, though some operations require the use of the mouse.

The figure shows the analyzer screen, whose elements are described below.



Click the pointed button to Zoom in

- Title with the text "UVI51 LOGIC ANALYZER:" followed by the name of the displayed LOG file.

- Upper buttons bar

The bar includes 6 buttons the user can click to:

Exit the analyzer. Equivalent to Esc key.

Create a file in HPGL format corresponding to the chronograms window. Equivalent to 'p' key.

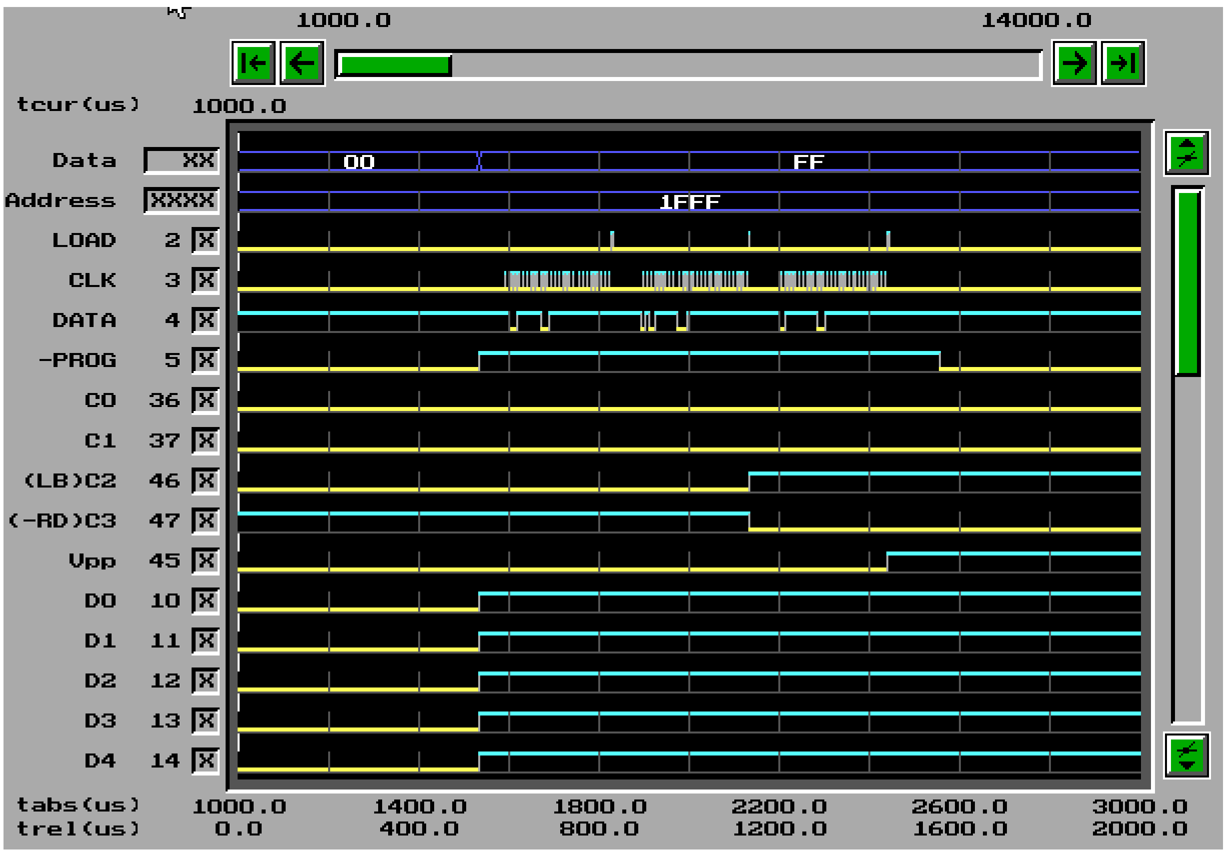
Show a help window. Equivalent to '?' key.

Zoom in time scale. Equivalent to '+' key.

Zoom out time scale. Equivalent to '-' key.

Find a pattern starting from cursor position to the end of the displayed block. When clicked keeping the shift key depressed, the search is extended beyond the displayed block. Equivalent to the 'f' and 'F' keys respectively. See section 7.3.

- Chronograms window:



This window shows node chronograms. It includes 2 shift bars. The horizontal bar is used to scroll through time axis. The vertical bar is used to select the group of nodes to be displayed.

Bars are shifted using the following buttons:

Time forward. Equivalent to the right arrow key.

Time backward. Equivalent to the left arrow key.

Go forward to the end of chronogram (maximum time). Equivalent to 'End' key.

Go back to the beginning of chronogram (minimum time). Equivalent to 'Home' key.

Scroll nodes up. Equivalent to 'PgUp' key.

Scroll nodes down. Equivalent to 'PgDn' key.

The bars can be shifted dragging with the mouse, or clicking the target position.

Node labels and numbers are shown in the left side, followed by their respective search patterns: X= don’t care, 0= low level, 1= high level. In the case of buses, the pattern is shown as an hex number: X= don’t care, ?= undefined, other = hex value. Section 7.3 indicates how to define and use search patterns.

The bottom of this window shows two time scales, one absolute indicating simulation time and the other relative to the logged data –zero for the first logged sample-.

A time cursor is also included. This cursor is shown as a vertical dashed line and has an associated time value shown on its upper side. The cursor can be moved using the following keys and/or button combinations:

Shift + → or Shift + right arrow key to move cursor to the right.

Shift + ← or Shift + left arrow key to move cursor to the left.

Shift + →| or Shift + ‘End’ key to move the cursor to the end of window.

Shift + |← or Shift + ‘Home’ key to move cursor to the beginning of window.

Cursor can also be dragged using the mouse or moved directly clicking the target position.

Cursor time is shown, by default, as absolute time. Clicking the cursor while keeping shift key depressed will reset cursor time to zero. This is useful for time measuring. If the same operation is performed but shift key is replaced by control key, cursor time recovers its absolute value.

Some operations, for example the display of very large and very active chronograms, can take a lot of time. The operation can be canceled using the ‘Del’ key.

The bottom of the screen shows a help on buttons. Simply move mouse pointer over a button –without clicking- to see the related help.

7.3. Finding patterns in chronograms

The task of locating a specific portion of chronograms can be hard, specially if a long simulation has been run. The find utility can help to locate any portion defined by a characteristic pattern. A pattern is a set of states (vector) related to a set of nodes. Before a search process is initiated, a search pattern has to be defined.

A pattern is defined assigning some node states to ‘0’ or ‘1’. That nodes whose state is *don’t care*, must be assigned an ‘X’ value. Node patterns change cyclically every time the user clicks them, according to the sequence ‘X, 0, 1’. The initial pattern is all *don’t cares*. If a bus has been defined, its search pattern changes automatically every time the user changes any of its node patterns.

The user can also modify a bus pattern directly, clicking on one of its hex digits. The clicked digit is replaced by a red square. Then, the user introduces the new value using the keyboard (0,...F). If the user depress the ‘X’ key, the four nodes that form the hex digit are marked as *don’t cares*. Any change in a bus pattern is updated automatically in the corresponding node patterns.

The question mark '?' placed in a bus pattern digit indicates that at least one of its nodes has been assigned as *don't care* and at least one has been assigned as '0' or '1'.

Once a search pattern is defined, the search process is initiated clicking the *magnifier* button or the 'f' key. The search starts from cursor position and extends to the end of the window. The cursor is moved to the first match found. For a search not limited to the present window keep the shift key depressed while clicking the button, or use the 'F' –uppercase- key.

If the cursor is already on a match, the new search begins after the present match is lost. This behavior makes successive searches easier.

APPENDIX A1: CNX Files Syntax

CNX files are ASCII files with two main blocks. The first block defines micro(s) while the second one defines external peripherals.

The expression *node number* is used throughout this section in a Spice-like way, though with some differences that must be considered:

- Node numbers must be numbers between 0 and 201. The use of correlative numbers starting with 2 is encouraged since this speeds up simulation.
- Numbers 0 and 1 have special meanings:
    - Node 0 → Connection to ground, equivalent to a permanent logic level 0.
    - Node 1 → Connection to +Vdd through a pull-up resistor. Its logic level is 1, unless connected to a 0 level (*Wired-AND* concept).

A1.1. Defining microcontrollers

Every micro in the system must be defined using a line with the following syntax:

Uprogram clockfreq xramsize romsize

where the initial U must be the first character in the line and has to be followed without blank spaces by a source file name. This is the program to be run by the micro. Three parameters follow indicating clock frequency –in MHz- and the size of external data and program memories –in bytes- respectively. The following ranges must be regarded:

1 <= clockfreq <= 100

16 <= xramsize <= 32768

32 <= romsize <= 8192

Once a micro has been defined, the user can add a set of lines in order to assign node numbers to micro ports. The syntax must be:

Pxy=N

with the meaning 'bit y of port x connected to node number N'. When the simulator finds this kind of line, it assumes a reference to the latest defined micro.

A1.2. Defining external peripherals

External peripherals are shown in the simulator system window (see section 6.1). The definition of a peripheral in the CNX file must include the position where the user want the peripheral shown. This position is specified by a cell number between 1 and 28. The following figure depicts the way cells are numbered.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32

The following peripherals are available:

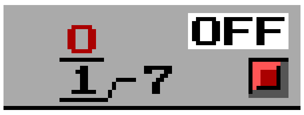
Switch

S<label> <common_node> <0_node> <1_node> <control>

If *control* > 0 then switch position is controlled manually by the user from simulator screen. If *control* < 0, switch position is controlled by the node whose number is the absolute value of *control*. Two examples follow:

SOFF 7 1 0 1

defines a switch with its terminals connected to:



common to node 7

idle to node 1 –pull up to Vdd-

active to node 0 –ground-

and to be shown in cell number 1.

Switch state –positioning- is accomplished clicking the little button at the bottom right of the cell. Label assignment –OFF in this example- eases switch identification.

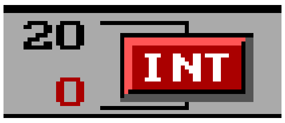
Cgate 7 1 0 –5

defines a switch whose state –positioning- is determined by node 5 logic level. This kind of switches -similar to relays or transmission gates- are not shown in the simulator system window. The current version imposes some restrictions to the use of these switches, in order to avoid convergence problems. Thus, the control node state must not depend on the positioning of other switch –not even the same switch-. In a typical application, the control node will be driven directly by a micro port.

Pushbutton (key)

K<label> <node_A> <node_B> <cell>

The pushbutton closes when depressed clicking the button shown in the simulator system window. Example:



KINT 20 0 1

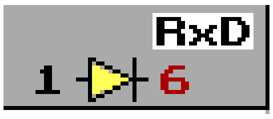
defines a pushbutton connected between node 20 and ground. The simulator shows the pushbutton in the cell number 1, where the user can depress it using the mouse.

Binary display (LED)

L<label> <anode_node> <cathode_node> <cell> [<color>]

The LED lights if anode_node is at logic level '1' and cathode_node at '0'. A lightened LED is shown painted with the given color. The default color is red. The following colors are available:

red	R
yellow	Y
green	G
blue	B



For example:

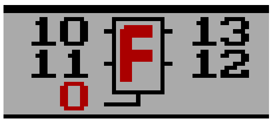
LRxD 1 6 8 Y

defines a LED whose anode is pulled up to Vdd through a resistor and whose cathode is connected to node number 6. The simulator will show it in the cell number 8. The figure depicts the LED lightened using yellow color.

Hex display

H <node_0(LSB)> ... <node_3(MSB)> <-enable> <cell>

This display is equivalent to a common cathode 7 segment display preceded by a BIN to HEX decoder. If the value applied to node_0 through node_3 inputs is restricted to the range 0 to 9, it can be used as a BCD to 7 segments decoder plus a common cathode display. If the -enable node is at 1 level, the display is blank. Thus, a multiplexed display can be simulated.



For example:

H 10 11 12 13 0 5

defines a display with its data inputs connected to nodes 10, ... , 13 and its -enable input permanently connected to ground. The display will be shown in cell number 5.

The -enable terminal is placed under de display. The 4 data inputs are placed on left and right sides beginning with the LSB in the left up corner, and following a counter-clock-wise sequence.

Logic Analyzer

Only one analyzer can be defined in a CNX file. The syntax is:

Afilename <tinitial> <tfinal>

where filename is the name of the file where the user wants the data logged. The parameters *tinitial* and *tfinal* indicate the simulator times when the data logging will begin and end respectively. The initial time is 0 by default. The final time is, by default, the end of simulation. The user can not indicate a final time if the initial time is omitted.

Data logging is carried out writing to a binary file. Data are registered using stamps. When any node state changes, a stamp is registered. Each stamp includes a time value and the state of all the nodes defined in the system. In order to avoid an unlimited file growth, the number of stamps is restricted to 100,000. The equivalent file size depends on the number of nodes in the system and is about 3 Mbytes. Would this limit be reached, the logging process is ended and the log file closed.

The analyzer definition can be followed by a set of lines assigning labels to nodes and defining buses.

Node labels

The user can assign labels to node numbers using the following syntax.

```
Nn  label
```

where *n* is the node number and *label* the name given to the node. For example:

```
N5 CLK
```

assigns the label CLK to node 5 signal. This label will be displayed by both the simulator and the analyzer replacing node number 5.

Buses

The user can define a bus as a sorted set of nodes using the syntax:

```
Blabel nn n0 n1 n2 ...
```

where label is the bus identifier. The parameter nn indicates de bus depth –how many nodes- and n0 n1 ... is a sorted list of nn node numbers beginning with the least significant one. For example:

```
BDATA 8 2 3 4 5 6 7 8 9
```

defines an 8-bit BUS named DATA whose LSB is node 2 and so on.

The relative position of lines defining node labels and buses within the CNX file is important because it determines their placements in the analyzer chronograms window.

APPENDIX A2: Customizing UVI-51

This appendix is not intended for novice users. The experienced users will find here an explanation on how to customize the batch files used to call the editor and the assembler.

The editor is called by the batch file _EDIT.BAT, the full path and name of the file to be edited being passed as a parameter. A typical BAT file for this purpose is:

```
@echo off

edit %1
```

that calls the MSDOS edit program. In this case, both edit.com and qbasic.exe must be accessible – their location directory must be included in MSDOS PATH-.

When the user requests UVI51 to assemble a source file, the batch file ASSEMBLE.BAT is called. The source file name without an extension is passed as a parameter. The standard ASSEMBLE.BAT file is

```
@echo off

uviens51 %1.src
```

whose second line call the UVI51 assembler.

In order to use Intel's ASM51 assembler, the ASSEMBLE.BAT file can be:

```
@echo off

copy %1.src/A %1.srb/B

asm51 %1.srb object(%1.obj) print(%1.lst)

hayerror %1

if errorlevel 1 goto goout

oh %1.obj to %1.hex

pause

:goout

del %1.srb
```

This file is customized for the peculiarities of ASM51. For other assemblers, the user is encouraged to consult the assembler manuals.

In any case, only the assemblers that generate LST and HEX files compatible with those generated by Intel's assembler can be used.

APPENDIX A3: Technical support

The author does not guarantee the correctness of this software, nor accept any responsibility for any loss derived of its use.

Furthermore, the author is not obligated to give present or future maintenance nor develop new software versions.

Nevertheless, the author will try to answer short and well defined questions about UVI51 usage, at the following mail address.

Alfredo del Río (UVI-51)
Dpto. Tecnología Electrónica
ETSII-Universidad de Vigo
36200 VIGO
SPAIN

and at the e-mail address:

ario@uvigo.es

indicating as subject UVI51.

The author encourage users to enclose the following information:

1) Requester’s name and occupation (University or College student, teacher, etc.).

2) E-mail address where the answer must be delivered (if different from sender).

3) Application where UVI51 is being or going to be used (teaching/learning at a specific level, R&D, etc.).

4) Requester’s opinion about UVI51: Assessment, errors, suggestions.

Thank you for using UVI51.

Vigo (Spain), December, 1999

Alfredo del Río.