

COPYRIGHT NOTICE

© Labcenter Electronics Ltd 1990-2007. All Rights Reserved.

The PROTEUS software programs (ISIS, PROSPICE and ARES) and their associated library files, data files and documentation are copyright © Labcenter Electronics Ltd. All rights reserved. You have bought a licence to use the software on one machine at any one time; you do not own the software. Unauthorized copying, lending, or re-distribution of the software or documentation in any manner constitutes breach of copyright. Software piracy is theft.

PROSPICE incorporates source code from Berkeley SPICE3F5 which is copyright © Regents of Berkeley University. Manufacturer's SPICE models included with the software are copyright of their respective originators.

WARNING

You may make a single copy of the software for backup purposes. However, you are warned that the software contains an encrypted serialization system. Any given copy of the software is therefore traceable to the master disk supplied with your licence.

PROTEUS also contains special code that will prevent more than one copy using a particular licence key on a network at any given time. Therefore, you must purchase a licence key for each copy that you want to run simultaneously.

DISCLAIMER

No warranties of any kind are made with respect to the contents of this software package, nor its fitness for any particular purpose. Neither Labcenter Electronics Ltd nor any of its employees or sub-contractors shall be liable for errors in the software, component libraries, simulator models or documentation, or for any direct, indirect or consequential damages or financial losses arising from the use of the package.

Users are particularly reminded that successful simulation of a design with the PROSPICE simulator does not prove conclusively that it will work when manufactured. It is always best to make a one off prototype before having large numbers of boards produced.

Manufacturers' SPICE models included with PROSPICE are supplied on an 'as-is' basis and neither Labcenter nor their originators make any warranty whatsoever as to their accuracy or functionality

INTRODUCTION

The aim of this guide is to take you through the process of entering a circuit of modest complexity in order to familiarise you with the techniques required to drive ISIS. The tutorial starts with the easiest topics such as placing and wiring up components, and then moves on to make use of the more sophisticated editing facilities, such as creating new library parts.

For those who want to see something quickly, `TRAFFIC.DSN` contains the completed tutorial circuit that we will use for the Interactive Tutorial and `ASIM1TUT.DSN` contains the tutorial circuit used in the Graph Based Tutorial. These circuits, together with over one hundred other samples, are seeded from the `SAMPLES` directory that by default is installed in:

```
C:\program files\Labcenter Electronics\Proteus 7 Professional\Samples
```

Specifically, the circuits above can be found in the `TUTORIALS` sub-directory on the above path.

It is important to note that a full reference manual is also included with the software and can be accessed from the *ISIS Help* command on the *Help* Menu in ISIS. Whereas this documentation provides a practical introduction to the software, the reference manual provides complete coverage of software functionality, including detailed discussion on more advanced functionality that is not covered in this guide.

You may also find our web based support forums useful for general Proteus enquiries and discussion :
<http://support.labcenter.co.uk>

Finally, if you still have questions or problems after consulting the reference manual please contact your local authorized distributor for technical support or mail us directly at support@labcenter.com quoting your customer number in the subject line of the email.

INTERACTIVE TUTORIAL

Overview

The purpose of this tutorial is to show you, through the creation of a simple schematic, how to conduct an interactive simulation using Proteus VSM. While we will concentrate on the use of *Active Components* and the debugging facilities of the ISIS Editor we will also look at the basics of laying out a schematic and general circuit management. Full coverage of these topics can be found in the ISIS Manual.

The circuit we will be using for the simulation is a pair of traffic lights connected to a PIC16F84 micro-controller as shown below.



Whilst we will be drawing the schematic from scratch, the completed version can be found as "Samples\Tutorials\Traffic.DSN" within your Proteus installation. Users who are familiar with the general operating procedures in ISIS may choose to use this ready made design and move on to the section on the micro-controller program. . Please note, however, that this design file contains a deliberate error - read on for more information.

If you are unfamiliar with ISIS, the interface and basic usage are discussed at length in both the ISIS Getting Started Documentation and in the online reference manual (Help Menu – ISIS Help). Although we will touch on these issues in the following section you should take the time to familiarise yourself with the program before proceeding.

Drawing the Circuit

Placing the Components:

We will start by placing two sets of traffic lights and a PIC16F84 on a new schematic layout. Start a fresh design, select the *Component Icon* (all the icons have both tool-tip text and context-sensitive help to assist in their use) and then left click on the letter 'P' above the *Object Selector* to launch the Library Browser. The Library Browser will now appear over the Editing Window (see Basic Schematic Entry in the ISIS Manual For more information).

Press the 'P' button on the keyboard and type in 'Traffic' in the Key words field and double click on the result to place the traffic lights in to the Object Selector. Do the same for the PIC16F84A.

Once you have selected both TRAFFIC LIGHTS and PIC16F84 into the design close the Library Browser and click left once on the PIC16F84 in the Object Selector

(this should highlight your selection and a preview of the component will appear in the Overview Window at the top right of the screen). Now left click on the Editing Window to place the component on the schematic - repeat the process to place two sets of traffic lights on the schematic.

Movement and Orientation:

We now have the building blocks on the schematic but the chances are they are not ideally positioned. To move a component, point the mouse over it and left click (this should highlight the component), then depress the left mouse button and drag (you should see the component outline 'follow' the mouse pointer) to the desired position. When you have the outline where you want release the left mouse button and the component will move to the specified position. Note that at this point the component is still highlighted - click on any empty area of the *Editing Window* to restore the component to it's normal status.

To orient a component, right click over it and then click on one of the rotation commands from the resulting context menu This will rotate the component through 90 degrees - repeat as necessary. Again, it is good practice to click on an empty area of the schematic when you are finished to restore the component to it's original state.

Set out the schematic in a sensible fashion (perhaps based on the sample given), moving and orientating the components as required. If you are having problems we advise you to work through the tutorial in the ISIS manual - ISIS Tutorial

For our purposes , we will ignore the 2D Graphics involved in the road junction and concentrate on creating a simulatable circuit - for those who are interested a full discourse on the Graphics capabilities of ISIS can be found here - 2D Graphics

Zooming and Snapping:

In order to wire up the schematic it is useful to be able to zoom into a specific area. Rolling the middle mouse button, hitting the F6 key or the *Zoom In* icon will zoom around the current position of the mouse, or, alternatively, holding down the SHIFT key and dragging a box with the left will zoom in on the contents of the dragged area. To zoom back out again, roll the mouse 'backwards', hit the F7 key or use the *Zoom Out* icon, or, should you wish to zoom out until you can see the entire design, hit the F8 key or use the mouse wheel to zoom out or in to the required area. Corresponding commands can be accessed through the View Menu.

More information on zooming and snapping can be found in The Editing Window in the ISIS manual.

Wiring Up:


The easiest way to wire a circuit is to use the Wire Auto Router option on the Tools Menu. Make sure that this is enabled (a tick should be visible to the left of the menu option). For more information see The Wire Auto Router in the ISIS manual. Zoom in to the PIC until all the pins are comfortably visible and then place the mouse pointer over the end of pin 6 (RB0 / INT). The mouse cursor should change to be a green pencil. This indicates that the mouse is at the correct position to connect a wire to this pin. Left click the mouse to start a wire and then move the mouse to the pin connected to the red light on one of the sets of Traffic Lights. When you get a green pencil cursor again left click to complete the connection. Repeat the process to wire up both sets of Traffic Lights as given on the sample circuit.

There are a couple of points worth mentioning about the wiring up process:

- You can wire up in any mode - ISIS is clever enough to determine what you are doing.
- When enabled, the Wire Autorouter will route around obstacles and generally find a sensible path between connections. This means that, as a general rule, all you need to do is left click at both end points and let ISIS take care of the path between them.
- ISIS will automatically pan the screen if you nudge the edge of the Editing Window while placing a wire. This means that you can zoom in to the most suitable level and, so long as you know the approximate position of the target component, simply nudge the screen over until it is in view. Alternatively, you can zoom in and out while placing wires (using the middle mouse button).

Finally, we have to wire pin 4 to a power terminal. Select the Terminal icon and highlight 'POWER' in the Object Selector. Now left click on a suitable spot and place the terminal. Select the appropriate orientation and wire the terminal to pin 4 using the same techniques as before.

More useful information on wiring up can be found at the following places in the ISIS reference manual:

-  At this point we recommend that you load the completed version of the circuit – this will avoid any confusion arising if the version you have drawn is in any way different from ours! Also, if you have not purchased the PIC Microprocessor Model Library, you must load the pre-prepared sample file in order to proceed.

Writing the Program**Source Listing**

For the purposes of our tutorial, we have prepared the following program which will enable the PIC to control the traffic lights. This program is provided in a file called TL.ASM and can be found in the "Samples\Tutorials" directory.

```

LIST      p=16F84 ; PIC16F844 is the target processor

#include "P16F84.INC" ; Include header file

CBLOCK 0x10 ; Temporary storage
    state
    11,12
ENDC

org      0 ; Start up vector.
goto    setports ; Go to start up code.

org      4 ; Interrupt vector.
halt    goto    halt ; Sit in endless loop and do nothing.

```

```

setports  clrw           ; Zero in to W.
          movwf  PORTA   ; Ensure PORTA is zero before we enable it.
          movwf  PORTB   ; Ensure PORTB is zero before we enable it.
          bsf    STATUS,RP0 ; Select Bank 1
          clrw           ; Mask for all bits as outputs.
          movwf  TRISB   ; Set TRISB register.
          bcf    STATUS,RP0 ; Reselect Bank 0.

initialise clrw           ; Initial state.
          movwf  state   ; Set it.

loop      call  getmask   ; Convert state to bitmask.
          movwf  PORTB   ; Write it to port.
          incf  state,W   ; Increment state in to W.
          andlw 0x04     ; Wrap it around.
          movwf  state   ; Put it back in to memory.
          call  wait     ; Wait :-)
          goto  loop     ; And loop :-)

          ; Function to return bitmask for output port for current state.
          ; The top nibble contains the bits for one set of lights and the
          ; lower nibble the bits for the other set. Bit 1 is red, 2 is amber
          ; and bit three is green. Bit four is not used.
getmask   movf    state,W ; Get state in to W.
          addwf  PCL,F   ; Add offset in W to PCL to calc. goto.
          retlw 0x41     ; state==0 is Green and Red.
          retlw 0x23     ; state==1 is Amber and Red/Amber
          retlw 0x14     ; state==3 is Red and Green
          retlw 0x32     ; state==4 is Red/Amber and Amber.

          ; Function using two loops to achieve a delay.
wait      movlw  5
          movwf  l1

w1        call  wait2
          decfsz l1
          goto  w1

          return

wait2     clrf   l2
w2        decfsz l2
          goto  w2
          return
          END

```

 There is, in fact, a deliberate mistake in the above code, but more of that later...

Attaching the Source File

The next stage is to attach the program to our design in order that we can successfully simulate its behaviour. We do this through the commands on the *Source Menu*. Go to the *Source Menu* now and select the *Add/Remove Source Files* Command. Click on the *New* button, , browse until you reach the "Samples\Tutorials" directory and select the `TL.ASM` file. Click open and the file should appear in the *Source Code Filename* drop down listbox.

We now need to select the code generation tool for the file. For our purposes the MPASM tool will suffice. This option should be available from the drop down listbox and so left clicking will select it in the usual fashion. (Note that if you are planning to use a new assembler or compiler for the first time, you will need to register it using the *Define Code Generation Tools* command).

Finally, it is necessary to specify which file the processor is to run. In our example this will be `tl.hex` (the hex file produced from MPASM subsequent to assembling `tl.asm`). To attach this file to the processor, right click on the schematic part for the PIC and then left click on the part. This will bring up the *Edit Component* dialogue form which contains a field for *Program File*. If it is not already specified as `tl.hex` either enter the path to the file manually or browse to the location of the file via the '?' button to the right of the field. Once you have specified the hex file to be run hit ok to exit the dialogue form

We have now attached the source file to the design and specified which *Code Generation Tool* will be used. A more detailed explanation on the Source Code Control System is available in the Proteus VSM online reference manual.

Debugging the Program

Simulating the Circuit

In order to simulate the circuit point the mouse over the *Play Button* on the animation panel at the bottom right of the screen and click left. The status bar should appear with the time that the animation has been active for. You should also notice that one of the traffic lights is green while the other is red and the logic state of the pins can be seen on the schematic. You will notice, however, that the traffic lights are not changing state. This is due to a deliberate bug we have introduced into the code. At this stage it would be appropriate to debug our program and try to isolate the problem.

Debugging Mode

In order to ensure that we are thorough in the debugging of the circuit we will stop the current simulation. Once you have done that you can start debugging by pressing `CTRL+F12`. Two popup windows should appear - one holding the current register values and the other holding displaying the source code for the program. Either of these can be activated from the *Debug Menu* along with a host of other informative windows. We also want to activate the *Watch Window* from which we can monitor the appropriate changes in the state variable. A full discussion of this feature takes place in the online reference manual.

Concentrating for now on the *Source* popup notice the red arrow at the far left. This, together with the highlighted line indicate the current position of the program counter. To set a breakpoint here hit the `ENTER` key (the breakpoint will always be set at the highlighted line). If we wanted to clear the breakpoint we could do so by hitting the `ENTER` key again but we will leave it set just now.

Setting a Breakpoint

Looking at the program it can be seen that it loops back on itself in a repeating cycle. It would therefore be a good idea to set a breakpoint at the beginning of this loop before we start. You can do this by highlighting the line (at address `000E`) with the mouse, and then pressing `F9`. Then press `F12` to set the program running. You should now see a message on the *Status Bar* indicating that a digital breakpoint has been reached and the Program Counter (PC) address. This should correspond with the address of the first breakpoint we have set.

A list of the debugging keys are given in the *Debug Menu* but for the most part we will be using `F11` to single step through the program. Hit the `F11` key now and notice that the red arrow at the left has moved down to the next instruction. What we have actually done is executed the 'clrw' instruction and then stopped. You can verify this by looking at the *W* register in the *Registers Window* and noticing that it has been cleared to zero.

What we now need to do is determine what we expect to happen on execution of the next instruction and then test to see if it actually does happen. For example, the next instruction should move the contents of the "w" register in to `PORT A` i.e. `Port A` should be cleared. Executing this instruction and checking the *Register Window* verifies that this is indeed the case. Continuing in this vein until you reach our second breakpoint you should notice that both ports have been cleared ready for output (as dictated by the `TRISB` register) and that the state variable has correctly been set to 0.

As this is a function call we have the option of *Stepping Over* the function (by hitting the F10 key) but for completeness we will step through each instruction. Hitting F11 here will jump us to the first executable line of the getmask function. Stepping forward we see that the move operation was successful and that we 'land' in the correct place for adding an offset of zero onto our lookup table. When we return to the main program therefore, we have the mask that we would expect. Single stepping further and writing the mask to the port we can see the correct result on the schematic. Single stepping again to increment the state is also successful as evidenced by the *Register Window* where the value for the W register is incremented by 1.

The single step takes us onto the instruction designed to wrap the state around to zero when it is incremented above 3. This, as can be seen from the *Watch Window*, is not performing as it should. The state should clearly be incremented to state 1 here in order for the mask to be set correctly on the next execution of the loop.

Finding the Bug

Closer investigation reveals that the problem is caused by **ANDING** with 4 instead of with 3. The states we want are 0,1,2, 3 and any of these when **AND**ed with 4 gives 0. This is why when running the simulation the state of the traffic lights does not change. The solution is simply to change the problem instruction to **AND** the state with 3 instead of 4. This means that the state will increment to 3 and when the W register is incremented to 4 the state will wrap around to 0. An alternative solution would be to simply test for the case when the 'W' register hits 4 and to reset it to zero.

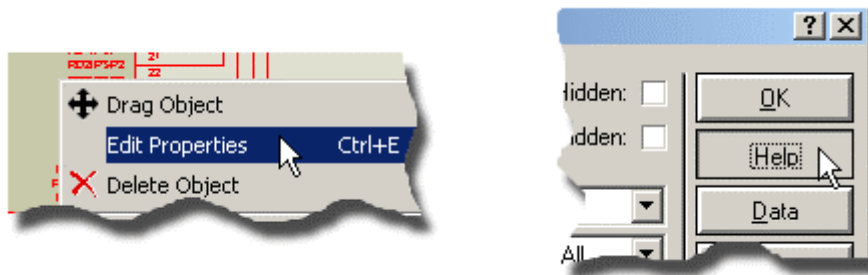
While this short example of the basic debugging techniques available in Proteus VSM illustrates the basics there is a lot of additional functionality available. Some of this functionality is only exposed when the firmware is written in a high level language and is discussed in more detail both in the following topics and in the online reference manual.

Taking Measurements

To take measurements in an interactive simulation you place and wire the appropriate virtual instrument onto the schematic prior to simulation. The instruments will then provide live feedback and interrogation of the virtual system during simulation. The full list of current available instruments is provided below.

- Oscilloscope
- Logic Analyser
- Counter/Timer
- Signal Generator
- Pattern Generator
- Dual Mode I2C Protocol Analyser
- Dual Mode SPI Protocol Analyser
- TTY/RS232 Virtual Terminal
- DC/AC Voltmeters and Ammeters
- Voltage and Current Probes

Each instrument has a help file associated with it which can be launched by right clicking on the component (once placed), selecting *Edit Properties* from the resulting context menu, and then hitting the Help button at the right of the resulting dialogue form. This help file explains all the modes of operation for the instrument and provides a useful reference source in correct operation.



Launching the Help File for a Virtual Instrument.

It is important to realise that all pulses, signals and waveforms propagate through the wires on the schematic – there is no cheating or hidden communication between components. This means that you can place and wire any instrument at any place on the schematic to interrogate the design at that point.

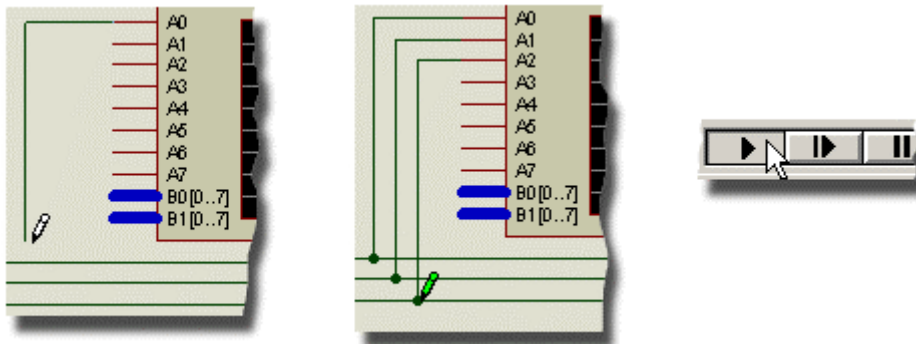
While not a particularly exciting example of this the following illustration covers the placement of the Logic Analyzer on the current schematic.

To place and use a Virtual Instrument

1. Select the *Instrument* icon and then select the instrument for placement from the *Object Selector*.
2. Left click on the *Editing Window* to enter placement mode, move the mouse to the desired location and left click again to commit placement.



3. Wire the pins of the instrument onto the wires of interest on the schematic.
4. Press the *Play* button on the *Animation Control Panel* to start the simulation and monitor data.



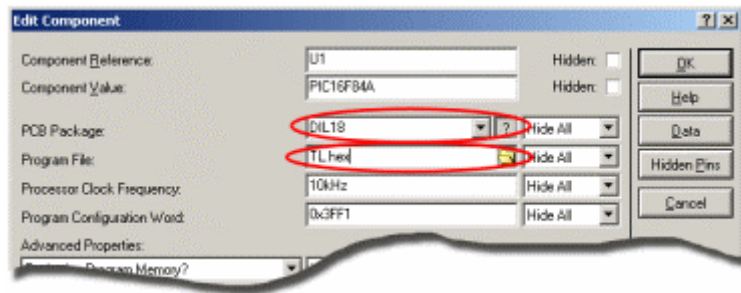
You can of course then drag our cursors to take measurements, print a timing diagram etc. – refer to the Logic Analyzer Help file for more information.

Writing Programs in C/C++

For simplicity the above tutorial is centered around a program written in assembly language. In practice, some if not most engineers today will use a compiler and write their firmware in C/C++. Proteus fully supports debugging of such programs via the object file output by the compiler. Indeed, as more information is provided with such files, the debugging functionality is typically enhanced when working in this way – see the topic on Advanced Debugging Techniques for an overview or the online reference manual for detailed information.

To Debug a C program using Proteus VSM :

1. Create a single folder for your project including ISIS schematic file and project source.
 2. Compile the source files, ensuring that your compiler produces an object file.
 3. Edit the processor component on the schematic and specify the clock frequency and set the program as the object file produced by the compiler.
 4. Start the simulation using the PAUSE button on the *Animation Control Panel*.
 5. Launch debug windows from the *Debug* menu, set breakpoints, single step etc. as detailed previously.
- i** Additional debugging facilities may be available (such as a variables window for example) depending on the information provided in the object file. Please see the topic on Advanced Debugging techniques below or the online reference manual for details.
 - i** The Clock Frequency is **always** taken from the schematic component. Wiring an oscillator to the clock pins consumes enormous CPU resources for no purpose.



Editing the CPU component and specifying clock frequency and program property.

Supported Object File Formats

When writing in C/C++ the object file that you supply to the schematic component depends both on the target processor and on the compiler that you are using to create the program. The following table outlines the supported formats (at the time of writing) for various target processors :

	COD	COFF	ELF/DWARF2	UBROF8	OMF51	HEX
PIC Families	✓	✓	x	✓	x	✓
Atmel AVR	x	✓	✓	✓	x	✓
Freescale HC11	x	x	x	✓	x	✓
ARM7/LPC2000	x	x	✓	✓	x	✓
8051	x	x	x	✓	✓	✓
Basic Stamp	N/A	N/A	N/A	N/A	N/A	N/A

- i** While the same binary (HEX) file as the physical chip is programmed with can be used within Proteus, the fact that no debugging information is supplied means that the program will be able to be simulated but not debugged. Not recommended if an alternative format is available.
- i** The limitations of the COD symbolic debug data format mean that the VSM debugging support for this product is limited to stepping through the machine code and watching specific memory locations Source level stepping and variable display are not supported. Typically compilers will also optionally produce COFF files which is a preferred option.
- i** The Basic Stamp processor contain an integrated tokeniser so all that is required is to supply the source file itself as the program property (no compilation).

The majority of the microcontroller sample designs supplied with the software are written in C/C++ and work as described above. While most compilers can be driven from the command line, and therefore could be configured to work with the source code control system within ISIS, the familiarity of the compilers IDE is maintained by following the simple steps outlined here.

ADVANCED DEBUGGING

Overview

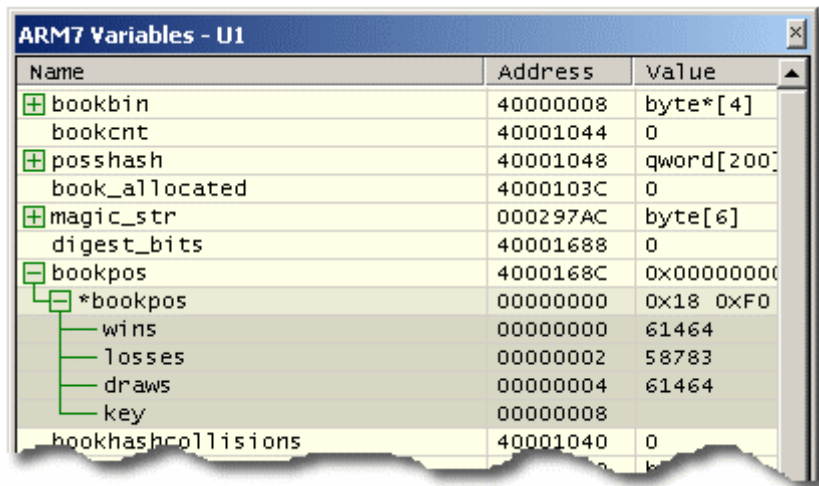
This section of the documentation discusses additional debugging techniques that can be used when working with microcontroller designs. In addition to the 'standard' techniques discussed in previous topics, Proteus VSM offers numerous advanced techniques and functionality to quickly help you fault find both in hardware and in firmware design. These are discussed in brief below – you can experiment with these techniques on the microcontroller sample designs pre-supplied with the software. The default path to the samples directory is:

```
c:\program files\labcenter electronics\proteus 7 professional\samples\
```

Debugging Windows

A variety of debugging windows are presented to the user via the *Debug* menu in ISIS during a simulation. These range from simple memory dumps to a register display, stack monitor and both watch and variables windows. Of particular interest is the watch window, which is visible during free running simulation and also allows you to specify watchpoint conditions (or conditional breakpoints).

Both the watch window and the variables windows can seamlessly handle compound types and will conveniently expand to display either SFR bit information or pointer, struct, enum information.



Name	Address	Value
bookbin	40000008	byte*[4]
bookcnt	40001044	0
posshash	40001048	qword[200]
book_allocated	4000103C	0
magic_str	000297AC	byte[6]
digest_bits	40001688	0
bookpos	4000168C	0x00000000
*bookpos	00000000	0x18 0xF0
wins	00000000	61464
losses	00000002	58783
draws	00000004	61464
key	00000008	
bookhashcollisions	40001040	0

Expansion of compound types in the variables window



More information on debugging windows together with some practical configuration examples can be found in the online reference manual (ISIS Help Menu – VSM Help).

Diagnostic Configuration

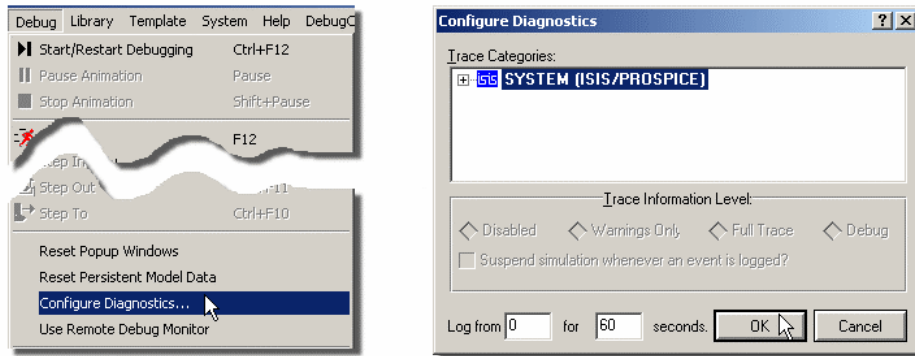
Proteus VSM includes extensive support for diagnostic tools or trace modes, which are invaluable in fault finding and verifying system operation. This is a mechanism by which all specified activity is logged during simulation and displayed on the simulation advisor in a textual reporting format. The format of the message includes the time the message was logged, a synopsis of the message, the component issuing the message and optionally further information in the form of context sensitive help. Think of it as a way to 'lift the lid' on a part and examine internal behaviour, providing traceability of data through the part and across the system.

Being a system level simulator, Proteus VSM includes diagnostic modes not only for microprocessors but also for appropriate peripheral models (LCD displays, I2C memories, temperature control devices etc.) and these trace modes can be enabled granularly via the configure diagnostic dialogue form. You can also control the level of messaging you want to see for each device. For example, you may want to examine full trace messaging on the onboard SPI peripheral of your microcontroller and also on the SPI EEPROM but may only be interested in receiving warning messages for other system components.

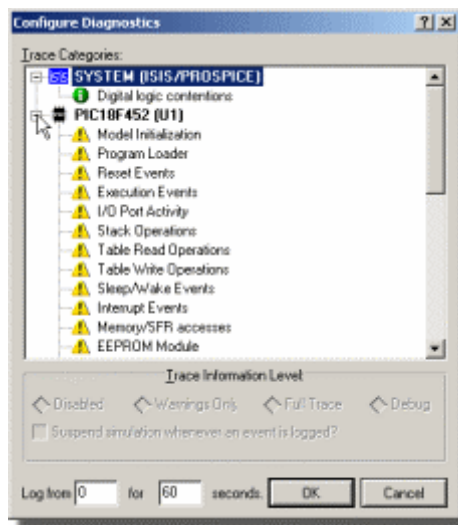
Configuring Diagnostics

To configure a simulation with diagnostics enabled:

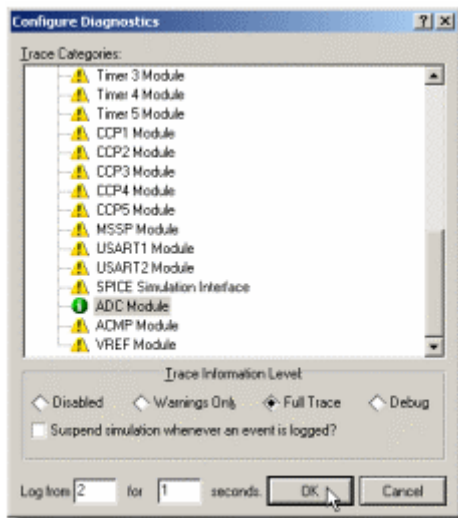
- Launch the Configure Diagnostics dialogue form from the Debug Menu.



- Expand the 'trees' in the dialogue form to find the item(s) for which you wish to enable diagnostics.



- Left click on the item of interest to select and then change the trace level to full trace. You can also set both an 'arm' time and a run time to control the time interval for which the diagnostics will be active.



- Repeat the process for any other items of interest and then exit the dialogue form.

When you run the simulation the configured trace diagnostics will become active at the arm time, run for the specified period and all results will be displayed on the simulation advisor.

- i Enabling trace diagnostics imposes a considerable load on the simulation. However, as they are typically used to identify obscure problems and for debugging purposes, the fact that the simulation will not run in realtime is not an issue.

The Simulation Advisor

The simulation advisor is the repository for all error, warning and diagnostics messages generated during a simulation run. It resides at the bottom of the ISIS application on the status bar next to the animation control panel. During a simulation run the status display will live update, indicating both the most severe type of message logged (errors, warnings, trace messages) and the number of such messages. You can launch the simulation advisor at any time during a simulation, or indeed after a simulation run (all messages are persistent!) by left clicking the mouse on the minimised display on the status bar.

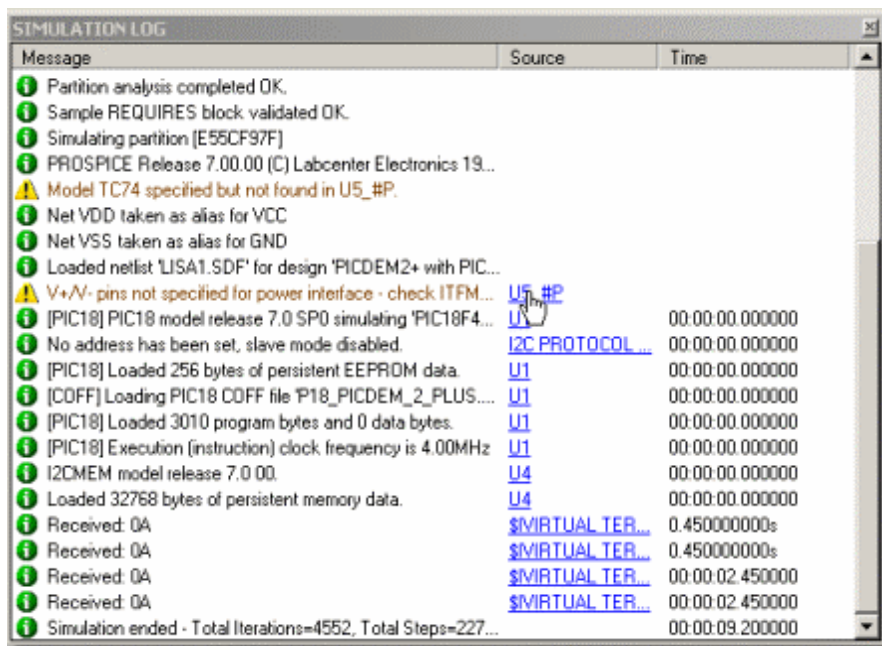


Launching the Simulation Advisor from the status bar.

The simulation advisor provides severity indicators beside every message as a visual indicator, navigation capabilities (both to a net and to a component on the schematic) where appropriate on messages and context sensitive help on common errors.

Navigation to a device with the Simulation Advisor

All messages originating from a physical component (pretty much everything except system messages) have an associated source column at the right hand side of the message. This serves to indicate the component on the schematic which generated the message and left clicking the mouse on the source link will minimise the simulation advisor and both zoom to and tag the source component.



Navigating to message source in the simulation advisor.

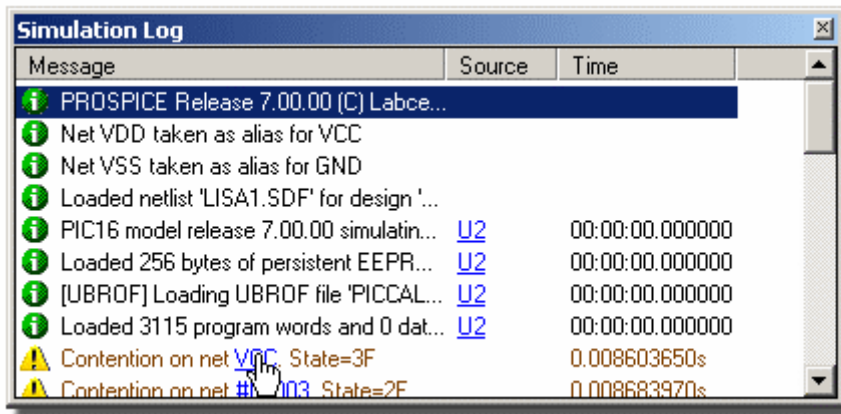
This is extremely useful, particularly in more complex designs, where you want to examine the hardware design around a device to resolve a problem.

Navigation to a net with the Simulation Advisor

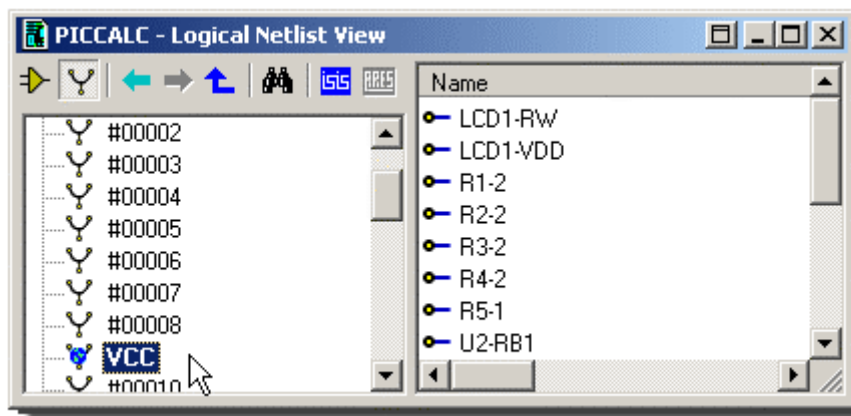
One of the most frustrating things about simulation disabled errors is that some problems (net contentions, SPICE singular matrices etc.) are relevant to a net and not a specific component and it is therefore extremely awkward to isolate the offending circuitry on the schematic. The simulation advisor simplifies this task by including 'hyperlinks' where applicable on messages, which when clicked allow you to navigate the design.

To navigate to a net from the simulation advisor:

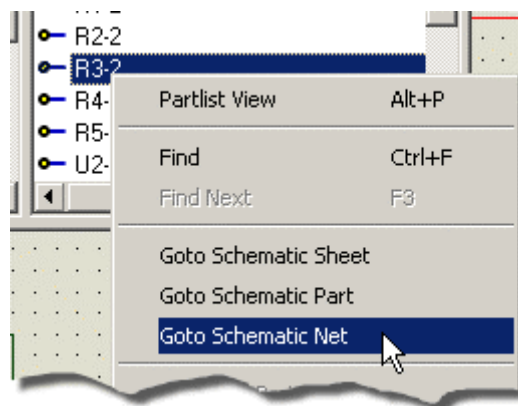
1. Click on the 'net link' contained in the message of interest within the simulation advisor.



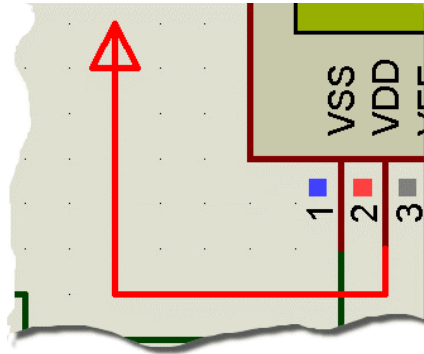
2. This will minimise the simulation advisor back to the status bar and launch the Design Explorer in ISIS, showing you a list of nets on the left hand pane and all the connections (pins) on the offending net in the right hand pane.



3. Right click on one of these pins and select the Goto Schematic Net option from the resulting context menu.



- The net in question will now be highlighted on the schematic.



- Note that this method is not foolproof as, where a schematic part is modelled via equivalent circuits (that is, the functionality of the part is modelled in schematic form by wiring together parts which already have models) and the problem net exists within the model, it is not possible to navigate to the net. However, while this and other parasitic cases exist, the technique above will work in most all normal situations and provides a powerful analysis tool in such cases.

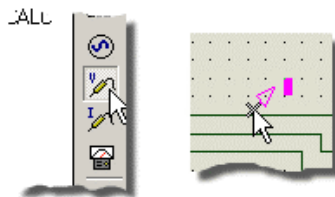
The design explorer is discussed in detail both in the ISIS tutorial and in the ISIS reference manual.

Hardware Breakpoints

A number of component objects are provided which trigger a suspension of the simulation when a particular circuit condition arises. These are especially useful when combined with the single stepping facilities, since the circuit can be simulated normally until a particular condition arises, and then single stepped in order to see exactly what happens next.

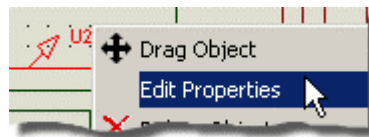
To set up a hardware breakpoint:

- Place a voltage probe on the wire (or bus) on which you wish to trigger the breakpoint.

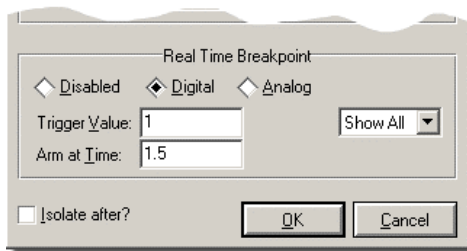


Adding a Probe to a wire.

- Right click on the probe and select Edit Properties from the resulting dialogue form.



- At the bottom of the dialogue form select either Digital or Analog according to the net on which the probe is attached and specify a trigger value. For a digital net and a single wire this will be either 1 or 0 corresponding to a logic high or a logic low respectively whereas for an analog net it will be a voltage value. You can also specify an arm time should you wish the breakpoint to become active after a period of time.



Setting the Trigger value and Arm Time in the Edit probe dialogue form.

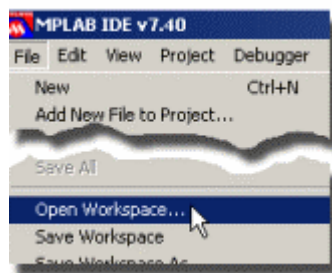
- Click OK to exit the dialogue form and then PLAY (or CTRL+F12) to run the simulation.

Working inside the MPLAB IDE

Labcenter and Microchip Technology® have collaborated extensively to provide a single environment solution to embedded design. It is possible to host the 'Proteus VSM Viewer' **inside** the MPLAB IDE, allowing you to more conveniently write, test and evolve your design. The following simplified guide assumes you have both MPLAB (V7.4 or later) and Proteus VSM Professional (Version 7 or later) installed on your computer. Note that this is a very basic guide to working with the viewer – you can launch the MPLAB help file from the help icon on the viewer itself for a more detailed discussion of measurements, debugging, etc. using MPLAB and Proteus.

Loading and Running a Proteus VSM simulation on the Explorer16 Evaluation Board :

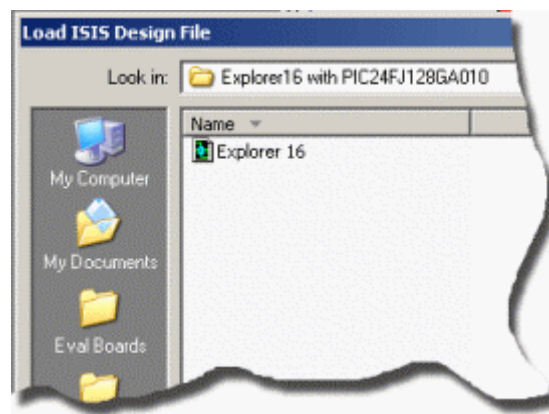
1. Launch the MPLAB IDE application, go to the File Menu and select the Open Workspace command. Navigate to the *Samples* directory of your Proteus installation.(by default this will be `c:\program files\Labcenter Electronics\Proteus 7 Professional\Samples\`) and then the 'VSM MPLAB Viewer\Eval Boards\Explorer16 with PIC24FJ128GA010' directory. Finally, open the `PIC24ExplDemo.mcw` workspace.



2. Go to the Debugger Menu in the MPLAB IDE, select the Select Tool command and then Proteus VSM. This configures MPLAB to use Proteus as the tool of choice for debugging.



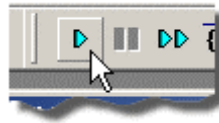
3. You should now see that the Proteus VSM Viewer has opened inside MPLAB. Use the Open Icon on the viewer and select the Explorer 16 schematic from the resulting file selector.



4. Select the *Build All* command from the *Project Menu* to ensure that our project code is both compiled and ready for simulation.
5. Now that we have the schematic and project we can start the simulation. Use the green button at the top of the MPLAB IDE to connect the Proteus simulation to MPLAB.



6. At this point the simulation is paused at time zero. Start the Simulation by clicking on the Play button near the top right of the MPLAB IDE. This will execute the program code and the VSM Viewer will show you the effects of your program on the design.



7. Use the red button at the top of the MPLAB IDE to disconnect the VSM Viewer from MPLAB and stop the simulation.



- i** Note that, because the PIC24 is a large part it has been placed on a second sheet on the schematic and the connection to and from the processor are made via terminals. You can navigate the sheets on the schematic by right clicking on a free area of the schematic and selecting the desired sheet.

GRAPH BASED TUTORIAL

Introduction

The purpose of this tutorial is to show you, by use of a simple amplifier circuit, how to perform a graph based simulation using PROTEUS VSM. It takes you, in a step-by-step fashion, through:

- Placing graphs, probes and generators.
- Performing the actual simulation.
- Using graphs to display results and take measurements.
- A survey of the some of the analysis types that are available.

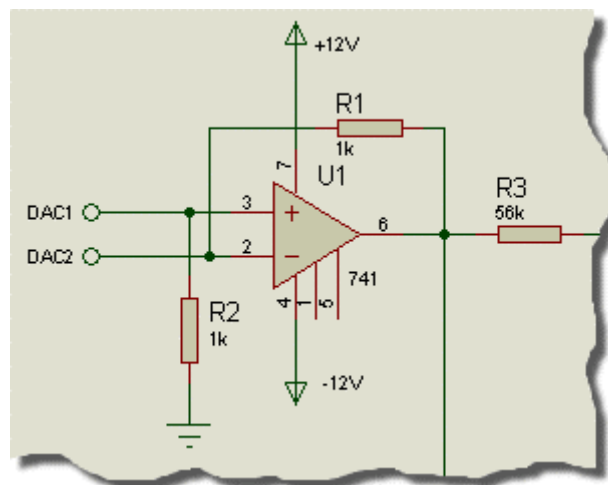
The tutorial does **not** cover the use of ISIS in its general sense - that is, procedures for placing components, wiring them up, tagging objects, etc. This is covered to some extent in the *Interactive Simulation Tutorial* and in much greater detail within the ISIS manual itself. If you have not already made yourself familiar with the use of ISIS then you must do so before attempting this tutorial.

We do strongly urge you to work right the way through this tutorial before you attempt to do your own graph based simulations: gaining a grasp of the concepts will make it much easier to absorb the material in the reference chapters and will save much time and frustration in the long term.

Getting Started

The circuit we are going to simulate is an audio amplifier based on a 741 op-amp, as shown overleaf. It shows the 741 in an unusual configuration, running from a single 5 volt supply. The feedback resistors, R3 and R4, set the gain of the stage to be about 10. The input bias components, R1, R2 and C1, set a false ground reference at the non-inverting input which is decoupled from the signal input.

As is normally the case, we shall perform a *transient analysis* on the circuit. This form of analysis is the most useful, giving a large amount of information about the circuit. Having completed the description of simulation with transient analysis, the other forms of analysis will be contrasted.



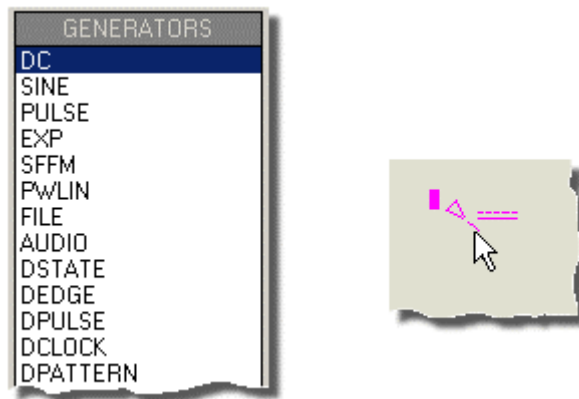
A Screenshot if a section from the ISISTUT.DSN

If you want, you can draw the circuit yourself, or you can load a ready made design file from "Samples\Tutorials\ASIMTUT1.DSN" within your Proteus installation. Whatever you choose, at this point ensure you have ISIS running and the circuit drawn.

Generators

To test the circuit, we need to provide it with a suitable input. We shall use a voltage source with a square wave output for our test signal. A generator object will be used to generate the required signal.

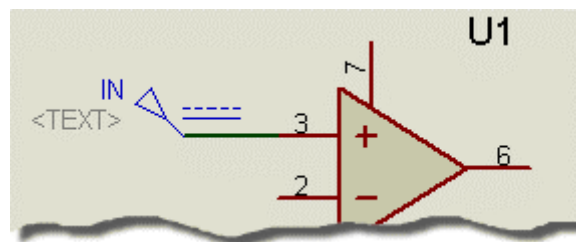
Click on the *Generator Icon* - the *Object Selector* then displays a list of the available generator types. For our simulation, we want a Pulse generator. Select the Pulse type, move the mouse over to the edit window, to the right of the IN terminal, and click left on the wire to place the generator.



Generator objects are like most other objects in ISIS; the same procedures for previewing and orienting the generator before placement and editing, moving, re-orienting or deleting the object after placement apply (see *Generators and Probes* in the online reference manual for more information).

As well as being dropped onto an existing wire, as we just did, generators may be placed on the sheet, and wired up in the normal manner. If you drag a generator off a wire, then ISIS assumes you want to detach it, and does not drag the wire along with it, as it would do for components.

Notice how the generator is automatically assigned a reference - the terminal name IN. Whenever a generator is wired up to an object (or placed directly on an existing wire) it is assigned the name of the net to which it is connected. If the net does not have a name, then the name of the nearest component pin is used by default.

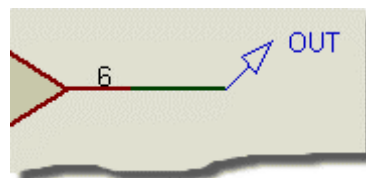


Finally, we must edit the generator to define the pulse shape that we want. To edit the generator, tag it with the right mouse button and then click left on it to access its Edit Generator dialogue form. Select the High Voltage field and set the value to 10mV. Also set the pulse width to 0.5s.

Select the OK button to accept the changes. For this circuit only one generator is needed, but there is no limit on the number which may be placed.

Probes

Having defined the input to our circuit using a generator, we must now place probes at the points we wish to monitor. We are obviously interested in the output, and the input after it has been biased is also a useful point to probe. If needs be, more probes can always be added at key points and the simulation repeated.



To place a probe, click left on the Voltage Probe icon (ensure you have not selected a current probe by accident - we shall come to these later). Probes can be placed onto wires, or placed and then wired, in the same manner as generators. Move the mouse over to the edit window, to the left of U1 pin 3, and click left to place the probe on the wire joining pin 3 to R1 and R2. Be sure to place the probe on the wire, as it cannot be placed on the pin itself. Notice the name it acquires is the name of the nearest device to which it is connected, with the pin name in brackets. Now place the second probe by clicking left just to the left of the OUT terminal, on the wire between the junction dot and the terminal pin.

Probe objects are like generators and most other objects in ISIS; the same procedures for previewing and orienting the probe before placement, and editing, moving, re-orienting or deleting the probe after placement apply (see the section on Probes for more information). Probes may be edited in order to change their reference labels. The

names assigned by default are fine in our case, but a useful tip when tagging probes is to aim for the tip of the probe, not the body or reference label.

Now that we have set up the circuit ready for simulation, we need to place a graph to display the results on.

Graphs

Graphs play an important part in simulation: they not only act as a display medium for results but actually define what simulations are carried out. By placing one or more graphs and indicating what sort of data you expect to see on the graph (digital, voltage, impedance, etc.) ISIS knows what type or types of simulations to perform and which parts of a circuit need to be included in the simulation. For a transient analysis we need an *Analogue* type graph. It is termed analogue rather than transient in order to distinguish it from the *Digital* graph type, which is used to display results from a digital analysis, which is really a specialised form of transient analysis. Both can be displayed against the same time axis using a *Mixed* graph.

To place a graph, select the *Graph* icon: the *Object Selector* displays a list of the available graph types. Select the *Analogue* type, move the mouse over to the edit window, click the left mouse button once and then drag out a rectangle of the appropriate size, clicking the mouse button again to place the graph.

Graphs behave like most objects in ISIS, though they do have a few subtleties. We will cover the features pertinent to the tutorial as they occur, but the reference chapter on graphs is well worth a read. You can tag a graph in the usual way with the mouse button, and then (using the left mouse button) drag one of the handles, or the graph as a whole, about to resize and/or reposition the graph.

We now need to add our generator and probes on to the graph. Each generator has a probe associated with it, so there is no need to place probes directly on generators to see the input wave forms. There are three ways of adding probes and generators to graphs:

- The first method is to tag each probe/generator in turn and drag it over the graph and release it - exactly as if we were repositioning the object. ISIS detects that you are trying to place the probe/generator over the graph, restores the probe/generator to its original position, and adds a trace to the graph with the same reference as that of the probe/generator. Traces may be associated with the left or right axes in an analogue graph, and probes/generators will add to the axis nearest the side they were dropped. Regardless of where you drop the probe/generator, the new trace is always added at the bottom of any existing traces.

The second and third method of adding probes/generators to a graph both use the *Add Trace* command on the graph menu; this command always adds probes to the current graph (when there is more than one graph, the current graph is the one currently selected on the *Graph* menu).

- If the *Add Trace* command is invoked without any tagged probes or generators, then the *Add Transient Trace* dialogue form is displayed, and a probe can be selected from a list of all probes in the design (including probes on other sheets).
- If there are tagged probes/generators, invoking the *Add Trace* command causes you to be prompted to *Quick Add* the tagged probes to the current graph; selecting the *No* option invokes the *Add Transient Trace* dialogue form as previously described. Selecting the *Yes* option adds all tagged probes/generators to the current graph in alphabetical order.

We will *Quick Add* our probes and the generator to the graph. Either tag the probes and generators individually, or, more quickly, drag a tag box around the entire circuit - the *Quick Add* feature will ignore all tagged objects other than probes and generators. Select the *Add Trace* option from the *Graph* menu and answer *Yes* to the prompt. The traces will appear on the graph (because there is only one graph, and it was the last used, it is deemed to be the *current* graph). At the moment, the traces consist of a name (on the left of the axis), and an empty data area (the main body of the graph). If the traces do not appear on the graph, then it is probably too small for ISIS to draw them in. Resize the graph, by tagging it and dragging a corner, to make it sufficiently big.

As it happens, our traces (having been placed in alphabetical order) have appeared in a reasonable order. We can however, shuffle the traces about. To do this, ensure the graph is **not** tagged, and click left over the name of a trace you want to move or edit. The trace is highlighted to show that it is tagged. You can now use the left mouse button to drag the trace up or down or to edit the trace (by clicking left without moving the mouse) and the right button to delete the trace (i.e. remove it from the graph). To untag all traces, click the left mouse button in a free area of the schematic, but **not** over a trace label (this would tag or delete the trace).

There is one final piece of setting-up to be done before we start the simulation, and this is to set the simulation run time. ISIS will simulate the circuit according to the end time on the x-scale of the graph, and for a new graph, this defaults to one second. For our purposes, we want the input square wave to be of fairly high audio frequency, say about 10kHz. This needs a total period of 100 μ s. Right click on the graph select *Edit Properties* from the resulting context menu to bring up its *Edit Transient Graph* dialogue form. This form has fields that allow you to title the

graph, specify its simulation start and stop times (these correspond to the left and right most values of the x axis), label the left and right axes (these are not displayed on *Digital* graphs) and also specifies general properties for the simulation run. All we need to change is the stop time from 1.00 down to 100 μ (you can literally type in 100 μ - ISIS will convert this to 100E-6) and select OK.

The design is now ready for simulation. At this point, it is probably worthwhile loading our version of the design ("Samples\Tutorials\ASIMTUT2.DSN") to avoid any problems during the actual simulation and subsequent sections. Alternatively, you may wish to continue with the circuit you have entered yourself, and only load the ASIMTUT2.DSN file if problems arise.

Simulation

To simulate the circuit, all you need do is invoke the *Simulate* command on the *Graph* menu (or use its keyboard short-cut: the space bar). The *Simulate* command causes the circuit to be simulated and the *current* graph (the one marked on the *Graph* menu) to be updated with the simulation results.

Do this now. The status bar indicates how far the simulation process has reached. When the simulation is complete, the graph is redrawn with the new data. For the current release of ISIS and simulator kernels, the start time of a graph is ignored - simulation always starts at time zero and runs until the stop time is reached or until the simulator reaches a quiescent state. You can abort a simulation mid-way through by pressing the `ESC` key.

The simulation advisor (beside the status bar at the bottom of the application) maintains any information about the last simulation run. You can view this log by left clicking the mouse over the simulation advisor or via the *View Log* command on the *Graph* menu. The simulation log of an analogue simulation rarely makes for exciting reading, unless warnings or errors were reported, in which case it is where you will find details of exactly what went wrong. In some cases, however, the simulation log provides useful information that is not easily available from the graph traces.



Launching the Simulation Advisor after a Simulation Run.

If you invoke the *Simulate* command a second time, you may notice something odd - no simulation occurs. This is because ISIS's partition management is clever enough to work out whether or not the part(s) of a design being probed by a particular graph have changed and thus it only performs a simulation when required. In terms of our simple circuit, nothing has changed, and therefore no simulation takes place. If for some reason you want to always re-simulate a graph, you can check the *Always Simulate* check box on the graphs *Edit Transient Analysis* dialogue form. If you are in doubt as to what was actually simulated, you can check the *Log Netlist* check box on the same dialogue form; this causes the simulator netlist to be included in the simulation log.

So the first simulation is complete. Looking at the traces on the graph, it's hard to see any detail. To check that the circuit is working as expected, we need to take some measurements...

Taking Measurements

A graph sitting on the schematic, alongside a circuit, is referred to as being *minimised*. To take timing measurements we must first *maximise* the graph. To do this, first ensure the graph is **not** tagged, and then click the left mouse button on the graph's title bar; the graph is redrawn in its own window. Along the top of the display, the menu bar is maintained. Below this, on the left side of the screen is an area in which the trace labels are displayed and to right of this are the traces themselves. At the bottom of the display, on the left is a toolbar, and to the right of this is a status area that displays cursor time/state information. As this is a new graph, and we have not yet taken any measurements, there are no cursors visible on the graph, and the status bar simply displays a title message.

The traces are colour coded, to match their respective labels. The OUT and U1 (POS IP) traces are clustered at the top of the display, whilst the IN trace lies along the bottom. To see the traces in more detail, we need to separate the IN trace from the other two. This can be achieved by using the left mouse button to drag the trace *label* to the right-hand side of the screen. This causes the right y-axis to appear, which is scaled separately from the left. The IN trace now seems much larger, but this is because ISIS has chosen a finer scaling for the right axis than the left. To clarify the graph, it is perhaps best to remove the IN trace altogether, as the U1 (POS IP) is just as useful. Click right on the IN label twice to delete it. The graph now reverts to a single, left hand side, y-axis.

We shall measure two quantities:

- The voltage gain of the circuit.

- The approximate fall time of the output.

These measurements are taken using the *Cursors*.

Each graph has two cursors, referred to as the *Reference* and *Primary* cursors. The reference cursor is displayed in red, and the primary in green. A cursor is always 'locked' to a trace, the trace a cursor is locked to being indicated by a small 'X' which 'rides' the waveform. A small mark on both the x- and y-axes will follow the position of the 'X' as it moves in order to facilitate accurate reading of the axes. If moved using the keyboard, a cursor will move to the next small division in the x-axis.

Let us start by placing the *Reference* cursor. The same keys/actions are used to access both the *Reference* and *Primary* cursors. Which is actually affected is selected by use of the CTRL key on the keyboard; the *Reference* cursor, as it is the least used of the two, is always accessed with the CTRL key (on the keyboard) pressed down. To place a cursor, all you need to do is point at the trace data (**not** the trace label - this is used for another purpose) you want to lock the cursor to, and click left. If the CTRL key is down, you will place (or move) the *Reference* cursor; if the CTRL key is not down, then you will place (or move) the *Primary* cursor. Whilst the mouse button (and the key for the *Reference* cursor) is held down, you can drag the cursor about. So, hold down (and keep down) the CTRL key, move the mouse pointer to the right hand side of the graph, above both traces, and press the left mouse button. The red *Reference* cursor appears. Drag the cursor (still with the CTRL key down) to about 70u or 80u on the x-axis. The title on the status bar is removed, and will now display the cursor time (in red, at the left) and the cursor voltage along with the name of the trace in question (at the right). It is the OUT trace that we want.

You can move a cursor in the X direction using the left and right cursor keys, and you can lock a cursor to the previous or next trace using the up and down cursor keys. The LEFT and RIGHT keys move the cursor to the left or right limits of the x-axis respectively. With the control key still down, try pressing the left and right arrow keys on the keyboard to move the *Reference* cursor along small divisions on the time axis.

Now place the *Primary* cursor on the OUT trace between 20u and 30u. The procedure is exactly the same as for the *Reference* cursor, above, except that you do not need to hold the CTRL key down. The time and the voltage (in green) for the primary cursor are now added to the status bar.

Also displayed are the differences in both time and voltage between the positions of the two cursors. The voltage difference should be a fraction above 100mV. The input pulse was 10mV high, so the amplifier has a voltage gain of 10. Note that the value is positive because the *Primary* cursor is above the *Reference* cursor - in delta read-outs the value is *Primary* minus *Reference*.

We can also measure the fall time using the relative time value by positioning the cursors either side of the falling edge of the output pulse. This may be done either by dragging with the mouse, or by using the cursor keys (don't forget the CTRL key for the *Reference* cursor). The *Primary* cursor should be just to the right of the curve, as it straightens out, and the *Reference* cursor should be at the corner at the start of the falling edge. You should find that the falling edge is a little under 10µs.

Using Current Probes

Now that we have finished with our measurements, we can return to the circuit - just close the graph window in the usual way, or for speed you can press ESC on the keyboard. We shall now use a current probe to examine the current flow around the feedback path, by measuring the current into R4.

Current probes are used in a similar manner to voltage probes, but with one important difference. A current probe needs to have a *direction* associated with it. Current probes work by effectively breaking a wire, and inserting themselves in the gap, so they need to know which way round to go. This is done simply by the way they are placed. In the default orientation (leaning to the right) a current probe measures current flow in a *horizontal* wire from left to right. To measure current flow in a vertical wire, the probe needs to be rotated through 90° or 270°. Placing a probe at the wrong angle is an error, that will be reported when the simulation is executed. If in doubt, look at the arrow in the symbol. This points in the direction of current flow.

Select a current probe by clicking on the *Current Probe* icon. Click left on the clockwise *Rotation* icon such that the arrow points downwards. Then place the probe on the vertical wire between the right hand side of R4 and U1 pin 6. Add the probe to the right hand side of the graph by tagging and dragging onto the right hand side of the minimised graph. The right side is a good choice for current probes because they are normally on a scale several orders of magnitude different than the voltage probes, so a separate axis is needed to display them in detail. At the moment no trace is drawn for the current probe. Press the space bar to re-simulate the graph, and the trace appears.

Even from the minimised graph, we can see that the current in the feedback loop follows closely the wave form at the output, as you would expect for an op-amp. The current changes between $10\mu\text{A}$ and 0 at the high and low parts of the trace respectively. If you wish, the graph may be maximised to examine the trace more closely.

Frequency Analysis

As well as transient analysis, there are several other analysis types available in analogue circuit simulation. They are all used in much the same way, with graphs, probes and generators, but they are all different variations on this theme. The next type of analysis that we shall consider is *Frequency* analysis. In frequency analysis, the x-axis becomes frequency (on a logarithmic scale), and both magnitude and phase of probed points may be displayed on the y-axes.

To perform a frequency analysis a *Frequency* graph is required. Click left on the *Graph* icon, to re-display the list of graph types in the object selector, and click on the *Frequency* graph type. Then place a graph on the schematic as before, dragging a box with the left mouse button. There is no need to remove the existing transient graph, but you may wish to do so in order to create some more space (click right twice to delete a graph).

Now to add the probes. We shall add both the voltage probes, `OUT` and `U1(POS IP)`. In a frequency graph, the two y-axes (left and right) have special meanings. The left y-axis is used to display the *magnitude* of the probed signal, and the right y-axis the *phase*. In order to see both, we must add the probes to both sides of the graph. Tag and drag the `OUT` probe onto the left of the graph, then drag it onto the right. Each trace has a separate colour as normal, but they both have the same name. Now tag and drag the `U1(POS IP)` probe onto the left side of the graph only.

Magnitude and phase values must both be specified with respect to some reference quantity. In ISIS this is done by specifying a *Reference Generator*. A reference generator always has an output of 0dB (1 volt) at 0° . Any existing generator may be specified as the reference generator. All the other generators in the circuit are ignored in a frequency analysis. To specify the IN generator as the reference in our circuit, simply tag and drag it onto the graph, as if you were adding it as a probe. ISIS assumes that, because it is a generator, you are adding it as the reference generator, and prints a message on status line confirming this. Make sure you have done this, or the simulation will not work correctly.

There is no need to edit the graph properties, as the frequency range chosen by default is fine for our purposes. However, if you do so (by pointing at the graph and pressing `CTRL-E`), you will see that the *Edit Frequency Graph* dialogue form is slightly different from the transient case. There is no need to label the axes, as their purpose is fixed, and there is a check box which enables the display of magnitude plots in dB or normal units. This option is best left set to dB, as the absolute values displayed otherwise will not be the actual values present in the circuit.

Now press the space bar (with the mouse over the frequency graph) to start the simulation. When it has finished, click left on the graph title bar to maximise it. Considering first the `OUT` magnitude trace, we can see the pass-band gain is just over 20dB (as expected), and the useable frequency range is about 50Hz to 20kHz. The cursors work in exactly the same manner as before - you may like to use the cursors to verify the above statement. The `OUT` phase trace shows the expected phase distortion at the extremes of the response, dropping to -90° just off the right of the graph, at the unity gain frequency. The high-pass filter effect of the input bias circuitry can be clearly seen if the `U1(POS IP)` magnitude trace is examined. Notice that the x-axis scale is logarithmic, and to read values from the axis it is best to use the cursors.

Swept Variable Analysis

It is possible with ISIS to see how the circuit is affected by changing some of the circuit parameters. There are two analysis types that enable you to do this - the *DC Sweep* and the *AC Sweep*. A *DC Sweep* graph displays a series of operating point values against the swept variable, while an *AC Sweep* graph displays a series of single point frequency analysis values, in magnitude and phase form like the *Frequency* graph.

As these forms of analysis are similar, we shall consider just one - a *DC Sweep*. The input bias resistors, `R1` and `R2`, are affected by the small current that flows into `U1`. To see how altering the value of both of these resistors affects the bias point, a *DC Sweep* is used.

To begin with place a *DC Sweep* graph on an unused space on the schematic. Then tag the `U1(POS IP)` probe and drag it onto the left of the graph. We need to set the sweep value, and this is done by editing the graph (point at it and press `CTRL-E`). The *Edit DC Sweep Graph* dialogue form includes fields to set the swept variable name, its start and ending values, and the number of steps taken in the sweep. We want to sweep the resistor values across a range of say $100\text{k}\Omega$ to $5\text{M}\Omega$, so set the *Start* field to `100k` and the *Stop* field to `5M`. Click on *OK* to accept the changes.

Of course, the resistors `R1` and `R2` need to be altered to make them swept, rather than the fixed values they already are. To do this, click right and then left on `R1` to edit it, and alter the *Value* field from `470k` to `X`. Note that the swept variable in the graph dialogue form was left at `X` as well. Click on `OK`, and repeat the editing on `R2` to set its value to `X`.

Now you can simulate the graph by pointing at it and pressing the space-bar. Then, by maximising the graph, you can see that the bias level reduces as the resistance of the bias chain increases. By $5M\Omega$ it is significantly altered. Of course, altering these resistors will also have an effect on the frequency response. We could have done an *AC Sweep* analysis at say 50Hz in order to see the effect on low frequencies.

Noise Analysis

The final form of analysis available is *Noise* analysis. In this form of analysis the simulator will consider the amount of thermal noise that each component will generate. All these noise contributions are then summed (having been squared) at each probed point in the circuit. The results are plotted against the noise bandwidth.

There are some important peculiarities to noise analysis:

- The simulation time is directly proportional to the number of voltage probes (and generators) in the circuit, since each one will be considered.
- Current probes have no meaning in noise analysis, and are ignored.
- A great deal of information is presented in the simulation log file.
- `PROSPICE` computes both input and output noise. To do the former, an input reference must be defined - this is done by dragging a generator onto the graph, as with a frequency reference. The input noise plot then shows the equivalent noise at the input for each output point probed.

To perform a noise analysis on our circuit, we must first restore `R1` and `R2` back to $470k\Omega$. Do this now. Then select a *Noise* graph type, and place a new graph on an unused area of the schematic. It is really only output noise we are interested in, so tag the `OUT` voltage probe and drag it onto the graph. As before, the default values for the simulation are fine for our needs, but you need to set the input reference to the input generator `IN`. The *Edit Noise Graph* dialogue form has the check box for displaying the results in dBs. If you use this option, then be aware that 0dB is considered to be 1 volt r.m.s. Click on *Cancel* to close the dialogue form.

Simulate the graph as before. When the graph is maximised, you can see that the values that result from this form of analysis are typically extremely small (pV in our case) as you might expect from a noise analysis of this type. But how do you track down sources of noise in your circuit? The answer lies in the simulation log. View the simulation log now, by pressing `CTRL+V`. Use the down arrow icon to move down past the operating point printout, and you should see a line of text that starts

```
Total Noise Contributions at...
```

This lists the individual noise contributions (across the entire frequency range) of each circuit element that produces noise. Most of the elements are in fact inside the op-amp, and are prefixed with `U1_`. If you select the *Log Spectral Contributions* option on the *Edit Noise Graph* dialogue form, then you will get even more log data, showing the contribution of each component at each spot frequency.