AVIDICY
Rapid Application development for the AVR Microcontroller

Demonstration Manual

AVIDICY is well featured development environment for Windows '95, '98, ME, 2000, XP and NT4.0.
The program, and its support files and example programs are
© Copyright Robin Abbott, 2003. <robin.abbott@fored.co.uk>.

The program may be installed onto the hard disk of only one Personal Computer, and must removed by deleting the executable file, and all the support files before installing onto a different computer.

Forest Electronic Developments
12 Buldowne Walk
Sway
Hampshire
SO41 6DU
Sales : +44 - (0)1590 - 681511

info@fored.co.uk

Or see the **Forest Electronic Developments** home page on the world wide web at the following URL:

http:\\www.fored.co.uk

AVIDICY Registration

This demonstration version of AVIDICY supplied for download from the web is a demonstration version which will not operate with any project other than the supplied demonstration project. However the full functionality of the application designer and simulator may be explored with the demonstration version.

Unlocking the demonstration version

To unlock the demonstration version and make available the full functionality of the program you need to send us the serial number for AVIDICY. In both programs this may be found under the Register menu option. Once we have accepted payment we will send you the registration key which will enable your program, we will also send the CD-ROM for the programs by normal post. The CD-ROM version does not require unlocking and will operate without a key. You can pay for the normal version or the professional version, we will send you the correct key to unlock normal, or professional version functionality.

You may pay by credit card (VISA or Mastercard), by email, or by fax, or send payment by post, or use bank transfer (ask us for details). The best payment method for credit card is to use our secure web site accessible through our web pages (below) - when you do this add the serial number to the comments box at the end of the checkout procedure.

The cost of enabling the program is shown on our web pages or write/email for details.

FED may be contacted at:

12 Buldowne Walk,
Sway
LYMINGTON
Hampshire
ENGLAND
SO41 6DU

Phone/Fax : 01590-681511
International : +44-1590-681511

Email : info@fored.co.uk

Web Site : http://www.fored.co.uk

Introduction

Welcome to AVIDICY, Forest Electronic Developments' series of rapid application development environments. It allows you to select software library components (*elements*) from a palette, set their parameters by drop down lists, check boxes and verified text entry. It will generate the main application, initialisation code and main loop automatically and considerably speeds the front end development of AVR projects.

AVIDICY includes the full functionality of the FED AVR C Compiler, and AVRDESIM - the FED development and simulation environment. AVIDICY can be configured not to use the application designer in which case it behaves exactly like the FED AVR C Compiler. AVIDICY can also load FED AVR C Compiler projects and handles them identically to the compiler.

This document is split into three, this first part includes a simple tutorial for the demonstration which shows how easy the system is to use. The second part is a full AVRDESIM tutorial for the simulator, and the third part is the AVIDICY reference.

You are recommended to run through the demonstration in detail – if you wish to upgrade to the full version you are also recommended to run through the examples in the full version introductory manual as these cover new toAVRs which are not applicable for the demonstration version.

C

It is expected that users of AVIDICY will be familiar with the C programming language. The AVIDICY CD ROM is supplied with an introductory manual to C "Learn to Use C with FED", it is recommended that this is read if unfamiliar with C.

Demonstration

This demonstration version is intended to show how to use AVIDICY, the simulator and the external device simulation. It is not possible to save files from the editor, open other projects, or compile any code other than the demonstration project. It is possible to experiment with the application designer, generate and compile the demonstration project and use the simulator with the project to the full including the wave window.

It is possible to upgrade the demonstration version to full functionality (either of the normal, or professional versions) by purchasing an activation key from FED. The CD-ROM will be sent by normal mail at the same time as the activation key.

Installation

The program is installed from CD-ROM or from the file downloaded from our web site. For the CD-ROM insert into the CD drive and an opening menu should come up. Alternatively run the program "SETUP.EXE" from the CD. For the web version download the program to a spare directory and run Setup.EXE.

It is strongly suggested that (at least initially) the program is installed in the default directory which will allow the example projects to operate correctly.

The manual is supplied as an Adobe Acrobat (PDF) Format file, a copy of Acrobat is supplied on CD-ROM and can be installed from the opening menu. The manual is duplicated in the help files which are accessible under the Help menu.

Running the program

Following installation there will be a new program group on the start menu, called AVIDICY. Double click the icon titled "AVIDICY" to start the program. The program will start and open the demonstration project. You may find that the project doesn't fit on the screen so you can use the menu option **Window | Arrange Neatly** to make it fit.

At this point you are recommended to run through the demonstration project in detail.

Demonstration Project – a simple LCD terminal.

In the example project we will look at the development and simulation of a complete program using AVIDICY.

The program we will look at is designed for a AT90S8515 processor – however note it may be converted to other AVR's with great ease. The application will undertake the following simple functionality:

1) It will have a serial interface which may be connected to a standard PC using a 9 pin socket.
2) It will have a hex keypad
3) It will have two LED's – red and green.
4) It will have a 2 line by 16 character LCD.

When a byte is received on the serial interface it will be shown on the LCD. As each byte is received it will be displayed in the next position on the display filling up each line at a time. When a key is pressed on the hex keypad the ASCII value of the key (from 0 to 9, or A to F) will be sent to the serial interface. Finally whenever a byte is received the Red LED will toggle (if it is on it will go off, if it is off it will go on), and whenever a key is pressed the green LED will toggle. The hex keypad will be debounced and keys will automatically repeat if they are held down.

By using the application designer we only need to write 10 lines of code to achieve this functionality.

We will simulate this device by using the real device simulation capability of AVIDICY.

The LCD and hex keypad attach to Port C and (LCD E signal is connected to Port D bit 2). The serial interface connects to port D using the internal hardware of the AVR (Port D0 and Port D1), the LED's are connected to Port B bits 0 and 1.

Starting the project

For the demonstration program much of the functionality has already been included. We will add some new elements to the application designer, look at the lines of C Code which need to be written and then run and simulate the code.

We assume that you have run AVIDICY as described above and are looking at the demonstration project.

The window on top is called the application designer. The application designer holds software elements in groups at the top of the window. A software *Element* is a library subroutine, or software component, which may be used within an application. The application designer allows software elements to be selected for use within the current application. The software elements are grouped by type.

There are other windows – ignore these for the present.

At the bottom of the application designer there is the element store – this holds the elements used on this project. In this case we have a keypad, an LCD, and the Timer 0 element. The Timer 0 element has been included automatically by the keypad element – it is used by the keypad element to time debounce and repeat periods. If you click each element in the element store you will see the element shown on the AVR and its connections to AVR pins. Note that the keypad and LCD share some pins on the AVR. You will also see the items on the parameters tab change as you click each in turn – the use of parameters will become clear as we run through the demonstration.

Using elements within the application

The first element that we will add is the serial interface. There are 3 asynchronous serial interface elements all under the Data tab. Select the Data tab and hover over an element with the mouse - a small help box will appear with the element name. Select the element called "Interrupt driven serial interface" by clicking it. The element icon looks like this:

Now right click the element and a pop up menu will appear. Select the menu option "Help on selected element" and read through the help file entry for this element. It probably won't all make sense at the moment.

Now you can add this element to the project by double clicking it, or by dragging it on to the picture of the AVR. Do this and the element will appear in the element store at the bottom of the application designer. A picture of the element in a box will appear on the drawing of the AVR.

Note that the element will have connected its pins to the AVR Port D, bits 0 and 1 which are the AVR hardware USART interface connections. They are called Rx and Tx. This element uses the same pins of the AVR whenever it is used.

Now we must set the parameters of the serial interface. Click the parameters tab and the parameters for this element will be displayed. The "Serial Bit Rate" should be selected to 19200. The "Use XON/XOFF flow control" option should be turned off – this is only used for controlling interface to an attached PC. The Receive and Transmit buffer sizes control how much RAM is reserved in the AVR for holding bytes received and due to be transmitted. Note that owing to the operation of the serial interface on most PC's the Receive Buffer Size should be set to 32. In our example we can leave these set to 32 and 8 bytes respectively.

Some software elements including this serial interface element allow the user to define software functions to be called automatically when events occur. An event is described in the Applications designer as an *Occurrence*. In this case the Occurrence is that a byte has been received on the serial interface. When a byte has been received we would like to display the received character. Click the Occurrences tab and a list of occurrences will be displayed, in this case there are two occurrences "TxFree" and "RxByte".

Click on the occurrence RxByte to select it. In the "Calls for Occurrence" box type DisplayChar and then click Add. The function DisplayChar will now be shown in the list of calls for this occurrence. Further functions could be added here if wished. Now whenever a byte is received the function DisplayChar will be called automatically – we will write the function DisplayChar later.

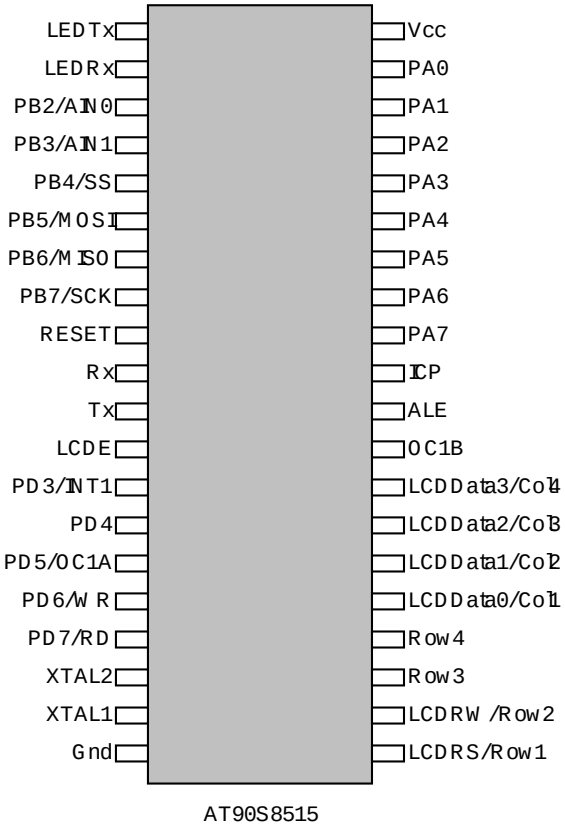
This completes the initialisation for the interrupt driven serial element.

Now we can add both LED outputs which are on pins B0 and B1. Click the Ports Tab at the top and add the "Port Driver" element by double clicking it, or by dragging to the AVR picture. The element looks like this:

Click the tab titled "Connect Pins". The drawing of the AVR will widen and you should be able to see the names of the element connections to the outside world on the element. In our example the LED output will be on pin PORTB, bit 0. Click pin RB0 of the AVR and it will turn red, now click the element pin called "Out0" and the element serial output will be connected to pin B0. Now we would like to name this pin LEDTx. Click the pin Out0 and both ends of the link will turn red to show that it is selected. In the Pin Name box enter “LEDTx” and the pin will be renamed. Click the Parameters tab and check that the initial value of the output is set to 0.

Now undertake the same procedure to connect another port driver to pin B1 – call this pin LEDRx.

We have now completed the work with the application designer - our application will include initialisation, code and data for all the main functions of the application. Examine the picture of the AVR - it should look like this, don't worry about the order of names on the pins of the AVR as it is dependant on the order of element selection.



You can print the AVR by right clicking the AVR graphic and using the *Print AVR* option of the pop menu. You can copy a picture of the AVR to the clipboard and paste into other applications by using the *Copy AVR* option of the same menu.

The elements which have been provided for you are the keypad and display. Click on the keypad element and the Occurences tab. Note that we have defined a function called KeyPressed which will be called whenever a key is pressed.

Generating the application for the first time

We have now selected the initial set of elements for the application, and the parameters, inputs, outputs, and occurrence calls have been defined, the application may be generated. To do this right click the AVR graphic and use the menu option *Generate Application* or use the small blue button at the top left of the application designer. The project window on the bottom left will show three files and a box titled "Compiler Options" will be shown, this allows the C Compiler options to be set. Click OK and the project will compile and assemble.

All being well there should be no errors. If there are errors then it is almost certainly because the LED pins have been wrongly named, or because the occurrences have the wrong names – check and correct if necessary.

If you cannot see the project window at the bottom right then use the menu option
Window | *Project - Demo.PC*

If you cannot see the items in any window properly then use the *Window* | *Arrange Neatly* menu option.

Now we would like to add some application specific code to the project. For the demonstration version this has already been done, however we can look at the code which has been created.

Double click the file “Demo_user.c” in the project window to open it. This file is produced automatically the first time that an application is generated (and is not generated again after that).

Examine the file.

There are two functions at the top of the file –UserInitialise and UserLoop. The blank template for these functions is automatically created when the program is first generated.

UserInitialise is called when the program starts and it holds one line of C Code:

```
LCDClear();
```

This clears the LCD display when the program starts.

UserLoop is called as part of the main application loop. It is empty in this example.

Look further down the file to this code:

```
//
// Occurrence - Byte received on serial interface
//
unsigned char LCDRow;
unsigned char LCDLine;

void DisplayChar()
{
 if (LCDRow++>=16)
 {
  LCDRow=1;
  LCDLine=!LCDLine;
  LCDPrintAt(0,LCDLine);
 }
 LCD(WaitRx());
 LEDRx=!LEDRx;
}
```

This is the code that we have written for the demonstration that displays a character when a byte is received on the interface. Recall that the function DisplayChar() is called automatically every time that a byte is received on the serial interface.

The variables LCDRow and LCDLine count the current print position on the display so that when 16 characters have been printed the next character is shown on the other line. The function LCDPrintAt(x,y) is a library function that sets the LCD print position to line y and character number x. The function LCD(n) prints the character n to the display. In this case the character is the next one received from the serial interface. The function WaitRx() returns the next character from the serial interface.

Finally note the code which toggles the LED. When the LED was named “LEDRx” the application generator automatically created three constants LEDRx for the port output pin, LEDRxDDR which may be set to 1 or 0 to set the pin to drive out, or as an input, and LEDRxPIN which is the value on the input. These constants are then used to toggle the LED.

Look at the other function which is called automatically whenever a key is pressed:

```
//
// Occurrence - A key has been pressed (or is repeating)
//

void KeyPressed()
{
 char tx=KP4Value+'0';
 if (KP4Value>=10) tx+='A'-'9'-1;
 // void AddTx(unsigned char V);
 // - Add byte V to Transmit buffer
 AddTx(tx);
 LEDTx=!LEDTx;
}
```

The variable KP4Value is defined within the library and is set to the number of the key pressed on the hex keypad. It will range from 0 to 15. This code converts the number from 0 to 15 to an ascii code from ‘0’ to ‘9’ or from ‘A’ to ‘F’ and transmits the byte on the serial port. It also toggles the Tx LED.

To look at the help file entry for any of these functions click on the function name (e.g. AddTx) and press Ctrl and F1.

We can also look at the autocode functions which help to write the code. (Bear in mind for the demonstration version that any changes to the code will be lost). Press Ctrl + End to go to the bottom of the file. Right click the edit window to bring up the menu and use the Autocode option (you could also press ALT and Enter). Use the Element Calls/Vars menu option to bring up a list of library functions for the elements that we are using. Select one with the mouse (e.g. LCDOnOff) to insert the function call and help information for that function. You can also use autocode to insert comment blocks, blank functions and loops.

Press Ctrl+F9 to create the application again.

This is now the completed application which can be programmed into a AT90S8515 and run directly. However we can also simulate it and look at the results on the waveform analyser.

Simulation

It is not the intention of the demonstration manual for AVIDICY to cover all the simulation capabilities of AVRDESIM which is covered in the later section. However we can check the operation of the program. It is possible to simulate with a stimulus file or with direct simulation of the external devices. Here we'll use simulation of the external devices.

Simulating with external devices

AVIDICY has the capability to simulate LED's, switches, LCD displays and a number of other devices which might be connected to the AVR.

Look at the debugging window. We have already added an LCD, both LED’s and the hex keypad to the example. We’ll look here at adding the terminal and then we can run the program for real.

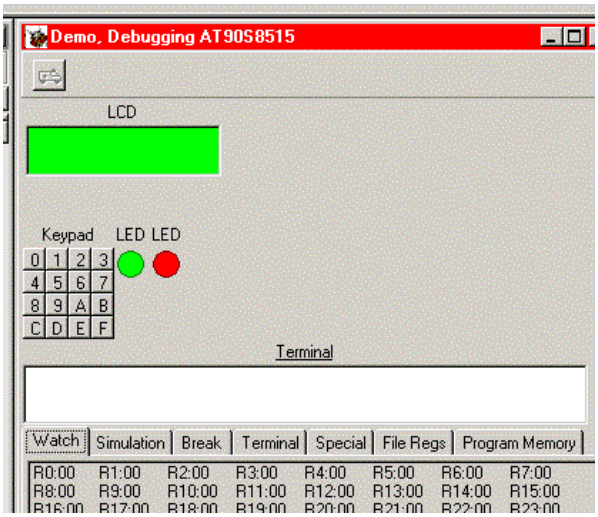
Use the **Simulate | Add External Device** menu option. A dialog box will come up. In the External Device type box select Terminal. There are a number of parameters and values which may be selected for each device. For the terminal we need to run at 19200bps. Look in the parameters box, select the Bit Rate option, and from the drop down box select 19200. Ignore the other two parameters.

Under the Connections box there is a list box called Pins with two entries "Terminal Rx" and "Terminal Tx". Select Terminal Rx and then use the Port drop down box to select Port D, use the Bit drop down box to select bit 1. Do the same for Terminal Tx, but connect to Port D bit 0. Note that the terminal Rx (receive pin) connects to the AVR Tx (transmit pin) and vice versa.

Click OK and note how the Terminal now appears on the debugging window.

We now have all 5 devices connected to the AVR. If you want to see how the other devices are connected then right click the description above the device and select the Edit option.

The top of the debugging window should look like this:



Note that the LED's are illuminated. This is because the simulator assumes all unconnected inputs to be logic 1. Look at the top of the screen, find the slider labelled "Update Rate". Move it to the extreme right, and then move it one notch to the left (this is the fastest rate which the simulator will run at whilst displaying variables as they update).

Now run the simulation by using the *Simulate* | *Run* menu option (you could also press F9, or use the button of the running man on the toolbar).

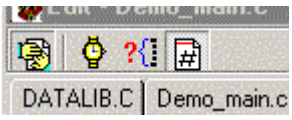
The LED's will go out. Click the terminal box (the cursor will appear) and type the letter A. Watch the LED go on, and the letter 'A' is shown on the LCD. Press 'B', the LED will go out. Press the keypad buttons with the mouse – watch the LED toggling and the values shown on the terminal. Hold a keypad button down and watch as the key auto-repeats – you will need to hold the key down for a little while.

If it doesn't appear to work. Have you moved the Update Rate slider ? Have you correctly assigned the RxByte occurrence of the terminal window to the DisplayChar routine ? Have you correctly assigned the Tx and Rx pins of the terminal (right click the title "Terminal" and use the edit menu option to change it).

You may also like to look at the LCDRow and LCDLine variables shown in the Watch tab of the debugging window. As you enter characters into the terminal window note how they are updated with the print position on the LCD.

Stop the program.

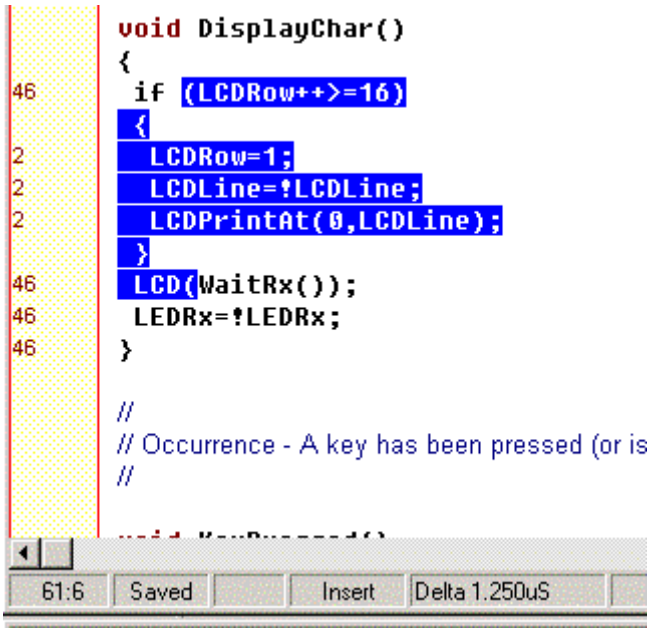
Now look at the information bar – the yellow bar to the left of the edit window. If the bar is not showing click the button to the top left of the edit window to turn it on (similarly click again to turn it off). Now the information bar can be set to show the address of each program line in the source, the time at which the line was last executed, or the number of times that it has been executed – click the buttons along the top of the edit window to select each option :



You can also determine how long a function or block of code takes to execute. Select a block of code – say the following block :

```
if (LCDRow++>=16)
 {
  LCDRow=1;
  LCDLine=!LCDLine;
  LCDPrintAt(0,LCDLine);
 }
 LCD(WaitRx());
 LEDRx=!LEDRx;
}
```

Now select the block of code with the mouse by clicking down before the void and dragging to the end of the function – you will see a box at the bottom of the edit window (titled "Delta") which will show the total time between the selected lines – in this case 1.25uS - the picture below shows the relevant part of the window:



You can drag over any area of code and the edit window will show the total time taken to execute that code provided that it has been simulated.

Using the waveform analyser

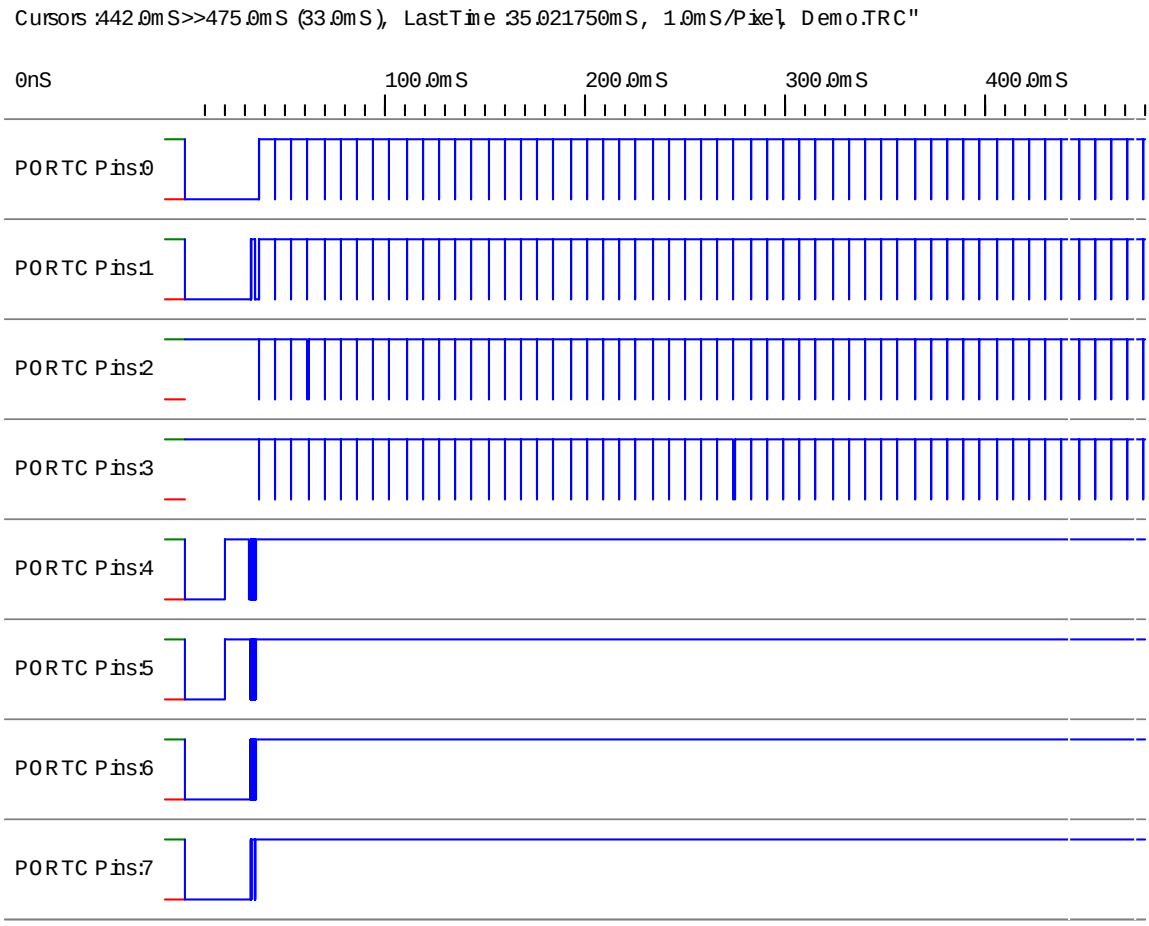
Now we want to run the program until time 500mS (half a second) and then stop to check the inputs and outputs. Select the debugging window and select the Break tab. Right click the window and select the *Add Breakpoint* option. (You could also use the *Simulate* | *Add Breakpoint menu* option.)

The Breakpoint Definition dialog box will be shown. Select "Break at Time", and then in the At Time box enter 500m, click the OK button to define the breakpoint. Reset the simulation using "Simulate | Reset Processor" or the blue button with a 0 on the tool bar.

Now run the program by using the *Simulate* | *Run* menu option (you could also press F9, or use the button of the running man on the toolbar). Wait until the breakpoint is hit at 500mS. The code will stop in the middle of an assembler statement – the cursor will stop on the line where the breakpoint was hit.

Use the *Tools* | *Examine Wave Window* menu option. The Wave Window "Define Trace Format" dialog box will be shown. In the Trace Name box select "Port C Pins", and then click the "Add as 8 line traces button". Resize the window to a comfortable size.

Press the F8 key repeatedly button to zoom out until the waveform occupies about 500mS (F7 zooms in if you've gone too far). The window can be copied to the clipboard or printed by using the options in the File menu for the Wave Window.



Look at the traces – at about 30mS you can see the LCD being initialised and then a repeating pattern on bits 0 to 3 as the rows of the keypad are strobed.

Return to the AVIDICY application. In the toolbar at the top of the main window slow the simulation down by by dragging the slider to the left a few notches. Reset the simulation and run it again. This time press a few keys in the terminal window. Wait until 500mS passes and examine the wave window again. You should be able to see a dramatic difference as characters are written to the LCD – note how the keypresses represented by long low periods on D0 to D3 result in data being written to the LCD.

You can experiment with the keypad. In the wave window use the Trace | Add Trace menu option to add PORT D bits 0 and 1 and PORT B bits 0 and 1 to the waveform window to see how these operate. You can move the vertical cursors around points of interest and press F6 to zoom up the wave between the cursors. Do this with PORT D bits 0 and 1 to watch serial information in detail.

Finally you can profile the performance of the program. Reset the program and run it until 500mS and use the Simulate | View Profile menu option. Many of the calls shown are internal to the compiler however you can see the KeyPressed and DisplayChar functions in the list (if you have pressed a key or sent a byte). Most of the time spent in these functions is within the library routines.

You are recommended to read through the simulator tutorial which is next within this manual to see some of the other simulation capabilities. However in the demonstration version, when it is not registered, you will not be able to add stimulus or injection files.

AVIDICY Registration

This version of AVIDICY supplied for download from the web is a demonstration version which will not operate with any project other than the supplied demonstration project. However the full functionality of the application designer and simulator may be explored with the demonstration version.

Unlocking the demonstration version

To unlock the demonstration version and make available the full functionality of the program you need to send us the serial number for AVIDICY. In both programs this may be found under the Register menu option. Once we have accepted payment we will send you the registration key which will enable your program, we will also send the CD-ROM for the programs by normal post. The CD-ROM version does not require unlocking and will operate without a key.

You may pay by credit card (VISA or Mastercard), by email, or by fax, or send payment by post, or use bank transfer (ask us for details). The best payment method for credit card is to use our secure web site accessible through our web pages (below).

The cost of enabling the program is shown on our web pages or write/email for details.

FED may be contacted at:

12 Buldowne Walk,
Sway

LYMINGTON
Hampshire
ENGLAND
SO41 6DU

Phone/Fax : 01590-681511
International : +44-1425-681511

Email : info@fored.co.uk

Web Site : http://www.fored.co.uk

FED AVR C - Simulator

Introduction

The FED AVR C allows you to simulate the complete compiled program within a single Windows environment. The simulator will be active as soon as the program has been successfully compiled. The simulator has its own menu option, the Debugging window is mainly used to display and control simulation effects, Input stimulus is provided through STI files (or for the professional version, the waveform generator). This section of the introductory manuals shows how to undertake various operations within the simulator.

The Debugging Window

The Debugging Window has seven main tabs, the watch tab, Simulation, Break, Special, Terminal and File Regs and Program Memory tabs. There are also two tabs which display the current content of file registers and the current contents of program memory (the latter is especially useful for applications which change program memory on an EEPROM device).

The Watch tab

The Watch tab shows the registers, the ports, and a list of memory locations which are being traced; their value is automatically updated whenever they change. Double click a register to bring up the Modify Memory Dialog Box which allows the value to be changed, similarly double click any item in the watch window to bring up the dialog box and change its value.

The watch tab has two windows. At the top is the list of variables and ports which are being examined. At the bottom is a listing of the stack. Double clicking a stack value will show the line in the main edit window to which the program will jump back when a return is executed at that location.

The Simulation Tab

The simulation tab has various controls on it which affect the simulation, for AVID-C the check memory access box will create an error when clicked if invalid memory is accessed. The Set PC button allows the Program Counter to be set anywhere in memory. The simulation frequency is set from the Application Designer.

The Break Tab

This tab shows the breakpoint list. Double click a breakpoint to edit it, press insert to create a new breakpoint, press delete to remove the selected breakpoint.

The Terminal tab

The terminal is a standard RS232 terminal which allows bytes to be sent to and received from application programs connected to the serial port. Right click the terminal window to bring up the options for it, or use the Module menu. The terminal window is selected by clicking the Terminal tab in the debugging window.

The options under the Module menu which refer to the terminal window are :

Trigger Dump. This option uses the Run Time Diagnostics facility.

Communications. This option brings up the Set Serial Port Parameters

Send File. This allows a file to be transmitted over the serial port.

The Special tab

The Special tab on the debugging window examines the special registers listed below, they will be updated at the same time as the watch variables.

The following registers are included amongst others:

Inputs The current values applied to the I/O pins of the processor, these will only be visible on the debugging window for the port registers if the DDR control registers for the port are set to 0's (for input).

Time in M The number of processor machine cycles which have been executed.

Watchdog An indication of what count has been reached by the watchdog timer.

EEPROM The memory dump of the EEPROM is shown for those devices which support EEPROM.

Other processor specific registers are also shown.

Controlling the Simulation

The update rate is a slider which is shown at the top of the IDE. At the left the value is 1. Almost at the right the value is 64000. This represents the number of cycles that the processor should execute before updating the watch variables. The higher the number then the faster the simulation. If the slider is set to the extreme right then the variables are not updated whilst running unless the “Update Now” button is pressed, or a simulation breakpoint is hit.

Also at the top of the IDE is the Stop on Errors Check Box. If enabled this will stop the simulation should an error occur during the simulation Run. If this Check Box is not selected, and if an error occurs, then errors will be printed in the Information Window, but the simulation will continue.

The simulation tab on the debugging window shows the other items used to control the simulation. The processor clock frequency is set in the Application Designer. The Change PC button allows the value of the PC to be set.

How to reset the Processor

To Reset the Processor then use the Reset Processor menu option under the Simulate Menu. Within a C program this command will then run the program until the first instruction in main(). To reset a C program without running to main then use the Simulate | Absolute Reset menu option which will reset to the first assembler instruction at address 0.

Zero the Time

To set the Time to Zero, which is useful when measuring Time taken for specific functions, then use the Zero-time menu option which does not affect the state of the Processor. The Time is set to Zero automatically when the Processor is Reset.

Run, Step, and Single Step the simulation.

To run the simulation then first the Program must be assembled successfully with no errors. There are a number of commands for running the simulation under the Simulate menu option.

1) **Run**. (F9) Once the Run button is clicked then the simulation will run until a breakpoint is hit, or the Cancel Button is pressed.

2) **Step within File Type**. (F7) The Step Command under the simulation Menu executes one Instruction in the current file type and updates the Debug Window. If the Program Counter is at a point in a C file then the program will run until the next line in a C file. If the Program Counter is at a point in an assembler file then the program will step to the next assembler instruction.

3) **Step Over**. (F8) The Step Over Command under the Simulate menu option executes a single Instruction, unless it is a Call Instruction in which case the Subroutine is executed and the Program stops once it has returned. This is useful for quickly stepping over the delay routines, or subroutines which are known to work correctly. Within a C file the Step Over command steps to the next program line within the program below the current line. Thus to step a C program both F7 and F8 should be used.

4) **Step to Next Instruction**. (F6) The Step to next instruction command under the simulation Menu executes one Instruction and will step to the next assembler instruction. When stepping a C program this instruction will allow stepping between the C program and the assembler listing

5) **Run to Cursor Line**. (F4) The program runs until the line on which the cursor is currently.

The professional version allows multiple projects to be simulated together. **The Simulate | Simulate all Projects Together** menu option is checkable. When selected all projects in a multi-project group will be simulated at the same time, if not selected then only the current project will be simulated.

Linking AVR pins to each other in the simulation

The professional version also allows AVR pins to be linked with pull up or pull down resistors. The **Simulate | Link AVR pins** menu option allows the pins of the devices in different projects in a multi-project group to be linked. The Connect AVR Pins Dialog box will be bought up.

Working with Watch Variables

The Watch tab on the debugging window shows the variables which can be examined whilst the program is running. Most actions concerned with watch variables are on the pop up menu of the watch list. Right click on the list to view the menu options.

To add a Watch Variable, then either use the Add Watch menu option under the Simulate Menu, or press the Insert key. This will bring up the Insert/Add Memory Watch Dialog Box (see below). One or more Watch Variables may be added in the Dialog Box by typing their addresses in label or numeric form separated by commas. The Browse button will display a list from which known variables may be selected. Alternatively right click on a variable name in the edit window and use the menu option to add a watch variable (or press F2).

If you are viewing a configuration register by its configuration address rather than memory address then use the Config check box.

To view a variable in its C form as declared in the C source then click the “Use C Definition” box in the Insert/Add Memory Watch Dialog Box. (Professional only).

To trace the watch variable (which allows it to be examined by the waveform analyser), then click the Trace button in the dialog box. The variable will be displayed with a * or a # to show that it is being traced.

To modify a variable in the debug window then select it and press enter.

To change the value of a variable in the debug window then double click it to bring up the Modify Memory Dialog Box.

To remove a Watch Variable, select it in the Debug Window and press the Delete key.

Insert/Add Memory Watch Dialog Box

The Insert/Add Memory Watch Dialog Box allows the user to enter one or more variables (memory locations) to the Debugging Window. If an item is selected in the Debug Window, then the variables will be inserted before the selected item. The Dialog Box contains the following items:

Use C Definition. Professional version only. This check box should be clicked with the C Compiler and AVIDICY if the variable is to examined using the C source as a guide to its content. The scope is also significant so within a block local variables will be shown ahead of global variables with the same name.

Trace. This check box should be clicked if the File Registers which are entered are to be traced during the run for examination in the waveform window (See Running the waveform analyser).

Config. This check box should be clicked if the address is a configuration (Input/Output) register, for example SPL, this will add $20 to the address typed into the address box as an offset into the I/O space.

Names or addresses. This edit box should contain one or more expressions for File Registers. If more than one expression is to be entered, then separate items with commas e.g. SPL,SPH,temp,temp+1 will insert four variables into the Debug Window.

Display As. This allows the display shown on the Debug Window to be configured. The display may be set to Bit, Byte, Word, or Long to display the value as a bit, as 1 byte, as 2 byte, or as a 4 byte value stored in memory. The Hex Dump options shows 16 bytes starting at the defined address. The bit option simply shows the value of a single bit of the address, the bit number to be examined should be entered in the edit box.

Display Format. This controls the format of the displayed value. Values may be displayed in binary, decimal or hex. Float and Zero terminated String format may also be used The signed button may be used to display decimal values in signed notation.

Pointer. If this box is clicked then the variable is taken to be a 2 byte pointer to an item defined by the other boxes. The value of the pointer, and the contents of RAM at that location are shown. If the pointer has its top bit set (bit 15) then the pointer is assumed to point to ROM, the top bit is reset to get the ROM address.

Browse. The Browse Button brings up a list of all File Register labels defined for this Program. One or more labels may be selected which will be added to the list of addresses in the Insert File Watch Dialog Box separated by commas when the >> Button is pressed.

Viewing and inspecting variables

The professional version includes a considerably improved variable inspector. The compiler saves information for the simulator to enable it to determine the scope and type of variables available within the program at all points within the source file.

When a variable is added to the watch window it is now possible to select it as being a C source variable. The “Use C Definition” check box in the top left of the Debug Watch dialog box will select a C variable. (Adding a variable to the debug watch window is most easily accomplished by pressing insert when the window is active, or using the Add Watch option of the menu which appears when the window is right clicked).

C Variables are shown with a small C symbol besides them:

For integer values the display format may be chosen as hex or decimal. The simulator will attempt to show the entire contents of the variable and dereference pointers. If the program moves to a new function or block where the variable name has a new meaning, then the C variable will always be shown as the value which is in scope.

Viewing Local Variables

The professional version allows the entire list of C local variables active at any time during the program to be shown by bringing up the Local Variable list. This is turned on or off by right clicking the watch window and using the “Inspect Local Variables” menu option. Users do not need to be aware of where local variables are held (stack or optimised memory), or how they are offset from the stack – the simulator determines all this automatically.

Inspector Windows

The professional version includes Inspector windows which allow the entire contents of a variable to be shown in detail. To inspect the value of a variable in detail on a pop up window then the **Simulate | Open Inspector** menu option. Alternatively position the cursor over a variable and use Shift F2, or use the right pop menu on the debug window when a watch variable is selected.

Inspector windows float over the main window and so are always available for examining values whatever the main program is doing. Inspector windows are not saved with the project and so must be opened as required during simulation.

How to change the program counter

To change the value of the PC click the Change PC button on the Simulation tab of the debug window, or click the PC box shown at the top of the Debug Window.

Alternatively use the Set PC to current line menu option on the Simulate menu. This will find the address of the line that the cursor is on within the edit window and set the PC to that line.

When one of the step or run instructions is used the program will now run from the new value of the PC.

Review your Program's Performance

The simulator maintains a count of the number of times a Label is called, and how long the Program spends before a Return Statement is executed, for up to 256 Subroutines. These Counts are Reset when the simulation is Reset. To review the Program's performance, then run the Program until a breakpoint, or the Cancel Button is pressed, then use the View Profile menu option under the Simulate Menu. This allows the timing data to be saved in a Text File which is an extension of .PRO. This File is then opened in the Editor to review Program performance.

Working with Breakpoints

The *Simulate* | *Add Breakpoints* menu option brings up the Breakpoints Dialog Box which allows the user to configure Program Breakpoints.

Breakpoints Dialog Box

The Breakpoints Dialog Box allows the user to add or change breakpoints in the program. The Dialog Box contains the following items:

Type. There are four types of breakpoint which may be selected
Unconditional Addressed : Enter an address and this breakpoint stops whenever the Program Counter reaches that address.
Conditional Addressed : Enter an address and a condition and this breakpoint stops whenever the Program Counter reaches that address, and the condition is non-zero.
Global Conditional : Enter an address and this breakpoint stops whenever the condition is true, regardless of the value of the Program Counter.
On Time : Enter a time (e.g. 1.0mS, 250uS, or 2.5S), and the program will stop when the simulation time equals the breakpoint time.

Address. This edit box allows the user to define the address for an unconditional breakpoint or an addressed conditional breakpoint. The address can be any valid AVRDE expression, which must be in the range of the processor’s program memory space. When the Break on Time button is clicked this box will be re-titled “Time”

Conditional Expression.. This edit box allows the user to define the Conditional Expression for a global conditional breakpoint or an addressed conditional breakpoint. The expression can be any valid AVRDE expression which will stop the Program when it evaluates to a Non-Zero result, eg. test ==9. This conditional expression will stop the Program when the value of the File Register test is 9.

Browse. The Browse button brings up a list of program labels. To add a label to the address edit box then select the label and double-click it, or select a label and press the >> button.

Animate. If this checkbox is clicked then the breakpoint is an animation breakpoint. An animation breakpoint does not stop the program, it simply causes the program to update all the watch variables when it is hit. This is most useful if the update slider is at the extreme right, in this event variables are only updated when an animation breakpoint is hit (or when the Update Now button is pressed). This allows the user to follow the value of watch variables when the program passes specific points.

OK. When the OK button is pressed, then all the breakpoints in the breakpoints list, and any breakpoint which is currently being edited, are set on the current Program.

How to set breakpoint on a line

Breakpoints are shown in the editor window with a blue circle for an enabled breakpoint, or a grey circle for a disabled breakpoint.

Using the mouse move to the left of the line on the editor window. It will change to a hand. Click to set a breakpoint which is shown with a blue circle. Click again to disable the breakpoint. Click once more to remove it.

Alternatively the *Simulate | Set Breakpoint on line* (or F5) menu command sets a breakpoint at the address which matches the position of the Cursor in an Edit Window if it exists. The simulator will report the address at which the breakpoint was set, or will report a failure if it cannot find an address.

If the cursor is on a line with a label then the breakpoint will be added with this label name, if there is no label on the line then it will be added as a numeric breakpoint.

Working with external devices

Any number of external devices may be simulated, and each type may be simulated any number of times - it is quite permissible to simulate 3 potentiometers and two LCD display modules for example. The following device types are available (see the simulator help file for more detail - use the **Help | Simulator Contents** menu option).

LED's, 7 Segment LED displays, LCD Modules, Potentiometers, Push Buttons, Hex Keypads, PWM voltmeters.

To add an external device then use the **Simulate | Add External Device** menu option.

To delete the device then click on the device name above it (the name will then be underlined), use the right mouse button to bring up the menu and use the delete option.

To edit the device (to change parameters or pin connections) then click on the device name above it (the name will then be underlined), use the right mouse button to bring up the menu and change the parameters in the box.

Working with Stimulus and Injection Files

To add or remove a Stimulus File (which contains Commands to set Port Inputs or File Variables to specific values) or Injection Files (which insert Hex Values in the Program), then use the project window.

To define a stimulus file for use in a program then add it to the project. Click the project window and press insert (or right click the window for a menu). Select one or more files in the file open dialog window (they normally have an extension of .STI). Now when the project type dialog box appears select Stimulus as the type of file.

Stimulus Files

Stimulus files allow the user to define data which appears as if it is present on the external pins of the Processor, or to change File Registers at specific times in the Program. Up to 16 Stimulus files may be included for any one simulation. Please note – input data to pins of the AVR should be applied to the PORT register NOT the PIN register (see examples below).

Stimulus files have the format described below:

A time should be given, which may be in the form of a number followed by an optional letter. If only a number is given, then the time is taken to be in nanoseconds. Alternatively, the letters U, M, or S may be placed directly after the number, in which case the time will be taken in microseconds, milliseconds or seconds, respectively. The time may be given at the start of a line with a command, or on a line on its own. Times do not have to be sequential and one file may define time and stimulus information which ends after another file. The time may be given in decimal format.

The following lines all specify the same time - 1mS:

```
0.001S
1m
1000u
1000000n
1000000
```

To trigger an event on a key press the command onkey is given with a letter or digit :

```
onkey 0
onkey A
```

Use onkey instead of a time, now when that key is pressed the following event will be triggered.

The commands available in a Stimulus file are described individually below, they allow a value to be applied to a Register, to a bit of a Register, or clocks or asynchronous data to be injected to a bit of a Register.
Each line contains a command as follows:

time Set the time for an event, follow with n,u,m or S for units all following events will occur at that time. If no units are provided then time is assumed in nS

```
e.g.    1000m
        200n
        1S
        35u
        1.050m
```

Note that times do not have to be in order - see the example at the end.

onkey Set the keyboard key to trigger a following event, note this must be on a line of its own

```
e.g.    onkey 0
        onkey 1
```

name=value Set specified file register defined by name to value. Note if = is given on its own then the last file register is assumed.

```
e.g.  1000m PORTA=76h        ; At 1000mS set file PORTA
                             ; to value 76Hex
1500m =0   ; At 1500mS set file PORTA
           ; to 0

onkey a
PORTA:0=1  ; Set PORT A bit 0 to 1 when key 'a' is pressed
```

name:bit=value Set specified bit of file register defined by name to value. Note if = is given on its own then the last file register and bit set is assumed. (Note if the last assignment was to a file register then the complete file register is assumed).

```
e.g.
          1000m PORTA:0=1    ; At 1000mS set file PORTA, bit 0
                             ; to 1
          1.5mS =0           ; At 1500mS set file PORTA, bit 0
                             ; to 0
          STATUS:3=0         ; At 1500mS set file status, bit 3
                             ; to 0
```

serialrate-name:bit=value

Inject value as an 8 bit asynchronous serial stream to the specified file (name), and bit. Note, if a name and bit value are not given then the last port and bit value are assumed. The rate is given in decimal immediately after the word serial.

```
e.g.
          10m serial9600-PORTC:0=65 ; At 10mS start the
                                    ; injection in
                                    ; serial format of the
                                    ; letter A
                                    ; to Port C, bit 0 at
                                    ; 9600bps
          200m serial2400-=66       ; At time 200mS start the
                                    ; injection of letter B to
                                    ; Port C bit 0, note the
                                    ; same port as
                                    ; that used last time is
                                    ; used.
                                    ; bit rate is 2400
```

clock-name:bit=low,high,cycles

Inject a clock to the specified file register and bit on that file register. low and high are the low time of the clock and the high time of the clock. cycles are the number of repeats of the clock. If cycles is 0, then the clock runs forever. The clock stays high when the number of cycles is complete

```
e.g.
        PORTB=0                       ; At time 0 set PORT B to
0

        100u                          ; Next event will occur at
                                      ; time 100uS
        clock-PORTB:0=25u,75u,10      ; This line generates 10
                                      ; clocks, which
```

```
                                                        ; are low for 25uS, and
                                                        ; high for 75uS
                                                        ; the clocks are driven to
                                                        ; PORT B, bit; 0
                                clock-PORTB:1=25u,75u,0 ; This clock runs forever
```

RTCC and Reset pins

To apply an input to the RTCC pin (the timer 0 input pin), or to apply a value to the reset pin, then instead of a port value use the labels PIN_RTCC or PIN_RESET. The lines below apply 100 20uS clocks to the RTCC (external timer 0) input pin, and then reset the processor at time 10mS, at time 12mS the reset is removed.

```
6m clock-PIN_RTCC=10u,10u,100
10m PIN_RESET=0
12m PIN_RESET=1
```

Analogue values

To apply an analogue input to any of the A/D input pins, then the same format may be used, however the value supplied is from 0 to 0FFFH, and represents the result of the A/D conversion. The higher values are only available for the 10 and 12 bit converters.

```
1m PORTD:0=080H   ; equivalent to 2.5V
2m PORTD:1=0C0H   ; equivalent to 3.75V
```

Injection Files (File format - injection files)

Injection Files allow the user to define hex data which is applied to a specified Register when the Program counter reaches a specified address. This is particularly useful in the situation where AVRDESIM does not fully emulate the behaviour of the real device, and a value can be injected as if the hardware of the real device was operating correctly.

To define an injection file for use in a program then add it to the project. Click the project window and press insert (or right click the window for a menu). Select one or more files in the file open dialog window (they normally have an extension of .HEX). Now when the project type dialog box appears select Injection as the type of file. Now it is possible to define the program address, and the memory address at which the data will be injected. Any normally valid AVRDE expression may be used.

The format of an Injection File is a number of hex bytes, each of which is on a line of its own, the hex data should be defined using digits 0 to 9 and letters A - F. Following the number then a comment may follow provided that it is separated from the number with a space. When the end of the file is reached then the file will be reset and data will be read from the start again. To aid layout of the file it is possible to insert blank lines where required.

For example:

```
00
FF
FE
00
```

This will insert the four bytes 00, FF, FE and 00 repeatedly and in turn.

Running the waveform analyser

The waveform analyser operates as a separate program, and allows the user to examine file registers which have been specially marked as for debugging. All of the ports are traced automatically. To set a watch variable as a traced variable then right click it and and use the *Modify Watch* menu option. Click the trace box, reset the simulation and run it - it will then be available in the waveform analyser.

Once the variables have been added to the Debug Window with the trace option set, and the simulation has been reset and then run, then the waveform analyser window may be used to examine them. Use the Tools | Waveform Trace menu option to bring up the analyser, which has its own help file and manual.

Waveform Generator – professional version only

The Waveform Generator is intended to allow users to design complex data and analogue patterns for injection to the pins of the device under simulation. It is a front end for the FED AVR and AVR development tools.

The wizard allows complex data patterns to be input to the AVR, clocks to be generated, or analogue waveforms to be generated for injection into the A/D converter inputs of the AVR. The waveform wizard allows a number of stimulus' to be stored together in one file. One of more of these files may be added to the list of project files and will then be included as simulation input when the program is simulated.

Within any of the FED development environments use the **Tools | Run Wave Generator** menu option to start the program. A new wave generator project will be created with the same name as the current project. The wave generator file will be added to the project window which will automatically include the stimulus when the program is simulated.

The Waveform Generator is described in more detail in the introduction to the professional version supplied with the software.

AVIDICY

AVR Microcontroller
C based visual development for Windows
Reference

AVIDICY is a well featured visual development environment for Windows ’95, ’98 and NT4.0.
The program, and its support files and example programs are
© Copyright Robin Abbott, 2000. <robin.abbott@fored.co.uk>.
The program may be installed onto the hard disk of only one Personal Computer, and must removed by deleting the executable file, and all the support files before installing onto a different computer.

Forest Electronic Developments
12 Buldowne Walk
Sway
HAMPSHIRE
SO41 6DU
Sales : +44 - (0)1590 - 681511

info@fored.co.uk

Or see the **Forest Electronic Developments** home page on the world wide web at the following URL:

http://www.fored.co.uk

Contents

Introduction
The Application Designer within AVIDICY
The Application Designer - concepts
Using the Application Designer with existing FED AVR C projects
Starting an application
Selecting AVR and oscillator frequency
Selecting and deleting elements
Setting Element parameters
Defining Occurrences
Connecting pins to the AVR
Listing public calls and variables
Generating the application for the first time
User code and the main loop - in brief
Structure of generated application
User defined code
Processors with multiple ROM and/or RAM pages
Macro Reference

Introduction

Welcome to AVIDICY, FED's unique drag & drop, point and click development of our popular AVR C development program. Please run through our tutorial first to see some of the capabilities of the program, this manual describes the operation of AVIDICY, and explains some of the procedures that you will need to follow to make best use of the program.

The Application Designer within AVIDICY

The Application Designer is the centre for selecting software library functions, connecting them to the AVR, and defining user defined functions which are called when events occur within the library function. The Application Designer has its own window within AVIDICY. To bring this window to the front use the **Window | Application Designer** menu option, or use the button which is at the extreme right hand side of the tool bar.

The application designer may be enabled or disabled by using the **Project | Use Application Designer** menu option. By default it will be enabled with a new project.

The Application Designer - concepts

The application designer holds software elements in groups at the top of the window. A software *Element* is a library subroutine, or software component, which may be used within an application. The application designer allows software elements to be selected for use within the current application.

Elements are extremely easy to generate using the *Element Editor*. Existing software components or libraries may be converted into software elements using the Editor which is described in its own manual.

When an element is selected for use in the application, the application designer will automatically hook it into the application, and set up the element according to the parameters selected within the designer. The application designer allows an element to be connected to any of the inputs or outputs of the AVR. Some elements have no connections to the pins of the AVR, others have a number of connections.

Some software elements also allow the user to define software functions to be called automatically when events occur. An event is described in the Applications designer as an *Occurrence*.

Examples of occurrences are the internal timers overflowing, a byte being received on a serial interface, for a seven segment multiplexed display changing digit.

Some elements include other elements automatically, and hook their functions automatically into occurrences of the element which they include. An example of this is the HMS element, which generates an occurrence every second, minutes, and hour that passes. This element automatically includes the Timer 0 element which drives the hardware for timer 0. The HMS element automatically hooks its functions into the overflow occurrence of Timer 0.

Using the Application Designer with existing FED AVR C projects

It is not recommended to use the application designer with existing projects until the user is very familiar with the operation of the designer.

By default the application designer will be enabled when an existing project is opened. It is recommended that it is disabled by using the **Project | Use Application designer** menu option when an existing project is opened.

Starting an application

A new project is started in AVIDICY by using the **Project | Open/New Project** menu option. A file open dialog box will shown, it is recommended that a new project is started in its own directory. As soon as the file name is selected and the OK button is pressed, the blank project will be created and the application designer is presented as the top window.

It is recommended that before any user specific code is written, that the application designer is used to create the minimum application. Then user code files can be inserted into the project window.

Selecting AVR and oscillator frequency

The first actions with a new project are to set up the AVR type and the oscillator frequency. The AVR type is selected with the "Change AVR" button under the element groups, the oscillator frequency may be selected from a list, or the exact value may be typed into the box. The oscillator frequency is entered in Hertz. Thus a 4MHz crystal should be entered as 4000000, however as this is a standard value it may be selected from the drop down box.

Selecting and deleting elements

To select an element then select the tab which holds the group from which you wish to make a selection. When the mouse is held over an element a help box will appear showing the name of the element and some information about it. The elements may be double clicked, dragged on to the AVR or the right button may be used to bring up a pop up menu which will allow the selected elements to be inserted.

Some elements are the unique, in that only one copy of the element may be used at any time. Other elements, such as the level detection elements which operate on a single pin of the AVR, may be selected as many times as necessary for the application. If an element is selected which may only be used once, then attempts to select additional copies of the element will fail. Elements which may only connect themselves to a certain pins of the AVR, such as the PWM element will automatically connect themselves to the AVR when they are selected.

The elements are all shown in the element store at the bottom of the Application Designer window. Elements may be selected from the element store by clicking it with the mouse, a small red box will be shown around the selected element. The selected element has its parameters, Occurrences, its list of public calls and variables, and the connections to the AVR shown in the main application designer window. The element may be defined within the application as described below.

Setting Element parameters

The parameters tab will display a list of parameters which may be changed for the selected element. Parameters may be selected as a number, or as an item from a list, or as a simple on or off selection by a check box. The full meaning of the parameters are described in the help file which may be shown for an element by right clicking it with a mouse, and using the help menu option.

Defining Occurrences

An *Occurrence* is an event which happens, which is detected by an element. It is possible to define user functions which will be called automatically when an occurrence happens.

To do this select the occurrence tab on the application designer for the selected element. The main list box will show a list of occurrences for the current element, click one of these occurrences to select it. Type the name for the assembler label at the start of the function into the box, and click the add button to include that function in the list of functions which will be called automatically when the occurrence happens.

To delete a call from the list of occurrences select the Occurrence in the main list box, then select the call to be deleted from the drop down box at the top of the window, and click the Remove button.

When an element, such as the HMS element, automatically loads another element, it may include its own functions within the list of calls which are to be executed when the occurrence. It is important that the user should not delete any of these calls, as they will prevent the element which hooks in the selected element from operating correctly.

Connecting pins to the AVR

Many elements may be connected to pins of the AVR. When an element is selected from the element store, it will be shown on the AVR outline, together with its inputs and outputs shown as pins alongside the elements within the main outline. To connect a pin from an element to a pin of the AVR, then click the pin of the AVR, and then the click the pin of the element to which it is to be connected. If the pin can be connected to that pin of the AVR then the outline will be re-drawn and the connection shown on the main AVR outline.

It is possible to connect more than one element to the same pin of the AVR, for example the LCD display driver allows the data lines to be shared with others inputs. If it is not possible to connect the element pin it to the AVR them an error box will be displayed explaining the reasons why the connection is not possible. For example the seven segment display driver element requires all segment drivers of the display to be connected to the same port, although any port may be selected for this functions.

To break a link then the connected pin of the AVR, or the element should be clicked to select the connection. The right mouse button may be used to bring up the menu, the break link option selected, or the break the link button on the Connect Pins tab used.

The AVR is shown in diagrammatic form within the application designer window. If the Connect Pins tab of the Application Designer is selected, then the outline of the AVR will be drawn in a wide form, allowing the pin names of the element, and the AVR to be read. If any of the other tabs is selected, then the outline will only show pin names on the AVR itself. The pin names shown on the AVR are those defined by Microchip, or when connected to an element, show the names of the element inputs/outputs.

The pins of the AVR will take on the name of the element pin to which they are connected. It is possible to use a customised name - for example the PortOut element sets a AVR pin to be an output and names it Out0, the second PortOut element output will be called Out1 etc.

It may be better and easier to understand to give them customised names - "StatusLED" and "MotorDrive" for example. The application designer will now create defines for these pins called StatusLEDPort and StatusLEDBit which can be used within the program.

To name the pins use the "Connect Pins" tab on the Application Designer. Select a pin on the current element, (or any pin of the AVR which is connected to an element) by clicking it. Then enter the name in Pin Name box on the Application Designer window - it will be shown on the AVR graphic as it is typed.

When an application is generated, the application designer will check that all element pins which must be connected are correctly connected to the AVR, if this is not the case then it will not be possible to go ahead and assemble the application. If the pin is named then variables will be generated related to the name which can be used in the program, for example if we call a pin TxSwtich and connect it to PORTD bit 2 then the following are created:

TxSwitchPort	The port used for the element (PORTD)
TxSwitchPin	The pin port (input) used for the element) (PIND)
TxSwitchBit	The bit number connected (2)
TxSwitch	A bit variable which is used for output for this element
TxSwitchDDR	A bit variable representing the DDR bit for this element's pin
TxSwitchPIN	A bit variable representing the input pin

To set the pin to output and drive a zero, the following code can be used:

```
TxSwitchDDR=1;
TxSwitch=0;
```

Listing public calls and variables

The interface tab of the application designer shows all the public calls and variables which may be used for the selected element. This information is of use to the programmer of the main code.

To automatically insert a blank call template for a C function for an element then press Alt and Enter whilst the cursor is in the Edit window, use the element Calls/Var option to bring up a list. Double click the required call.

Generating the application for the first time

Once the initial set of elements for the application have been selected, and the parameters, inputs, outputs, and occurrence calls have been defined, the application may be generated. To do this use the pop-up menu option, **Generate Application** within the application designer window, or use the main menu option **Assemble | Generate application**, or press Ctrl+F9, or use the tool bar button.

The application designer will automatically generate three or four files and then go on to compile the application (this is described in the Compiler reference manual).

The first of these is the application designer header file. It has the name ProjectName_Auto.h where ProjectName is the name selected for the project when it was first opened. the second is the main application file named ProjectName_Main.c, which includes the initialisation code, and the main interrupt routines. It should be included in all files added to the project.

The next file is the user file. It has the name ProjectName_User.Inc. Initially this simply contains blank functions for initialisation, interrupts and main loop. Blank functions are created for occurrence calls. This is the only automatically generated file which may be modified as it will only be generated once. The functions are described in detail below.

The final file is the library file which includes all the library routines from the various library files, they are all gathered together here. It has the name ProjectName_Lib.Inc. It will not be generated if the elements use no code for library files.

The include, main and library files should not be altered at all, as they will be completely rewritten every time that the Generate Application option is used.

These files are automatically included within the project window so they will be assembled in order. Additional C files may be added or inserted freely.

After the application designer has generated the main files, and included them within the project window, the Compiler Options dialog box will be bought up allowing the project to be compiled. All of the C options may be set with this box as described in the main C reference, or simply click the Help button.

User code and the main loop - in brief

All user code should be written in the ProjectName_User file, or additional files. Additional code may be added into the project in c files. To add additional files to the project window then click on a file and press insert, select a file and it will then be included before the file which was selected. Alternatively de-select all files and press insert to add a new file at the end. It is recommended that files be inserted between the _User and _Lib files. If the file is not found then it will be created. Further details are in the AVRDESIM example and on-line reference.

The user file contains user generated code for initialisation, interrupts, and the main loop. The main loop is code which the AVR executes in turn - testing for occurrences, calling any attached sub-routines, and calling any library routines which need to be called regularly. User defined code may be attached to the main loop, but should return to the main loop to continue the main functions of the program.

Attaching user defined code to the ProjectName_User file is straightforward. Double click the file in the Project Window and it will open in the Edit window. The various are described in comments in the file - and in detail below.

Structure of generated application

The Application Designer generated program follows the flow shown below:

```
void main()
{
                    Initialise essential registers
                                v
                                v
                        Initialise Elements
                                v
                                v
    Set I/O pins to correct state using TRIS statements
                                v
                                v
Initialise Elements - code added after TRIS registers are set up
                                v
                                v
                        Enable Interrupts
                                v
                                v
                  Call function UserInitialise()          // ANY USER INITIALISATION CODE
                                v
                                v
 while(1)                                        // MainLoop
 {                              v
                     For Each Occurrence
                                  Test Occurrence Flag
                                v
                                v
          Call functions associated with occurrence
                                v
                                v
Call any Element functions which are to be called regularly
                                v
                                v
                   Call function UserLoop()           // USER CODE HERE
                                v
                                v
                                v
 }
}
```

User defined code

The Application Designer automatically generates a file called ProjectName_User.ASM. This file is only generated once when the Generate Application option is first used, and thereafter may be modified at will. The blank generated file is shown below:

```
//
// This file includes all user definable routines. It may be changed at will as
// it will not be regenerated once the application has been generated for the
// first time.
//

//*****************************************************************************
//
// Insert your interrupt handling functions if required here.
//
// e.g.
//
// void InterruptOVF0
// {
```

```
//  // Your code here
// }


//*****************************************************************************
//
// Insert your initialisation code if required here.
// Note that when this routine is called Interrupts will not be enabled - the
// Application Designer will enable them before the main loop
//

void UserInitialise()
{
}

//*****************************************************************************
//
// Insert your main loop code if required here. This routine will be called
// as part of the main loop code
//

void UserLoop()
{
}

//
// User occurrence code
//
```

There are three functions here :

void UserInitialise()

Any user defined initialisation code should be written in here, it will be called immediately before the main loop.

User Interrupts

Very few elements use interrupts, most rely on detecting the interrupt flag. Those elements which use interrupts (such as the serial driven interrupt routines) supply interrupt code as part of the library function, these interrupts cannot be used by user code. However all other user interrupts may be inserted in the _User file as shown in the main C manual.

void UserLoop()

The main loop undertakes any processing necessary for occurrences and then jumps to UserLoop. Any user defined routines which need to be called regularly can be included here.

FED AVR C Compiler

The C Compiler used by AVIDICY is identical to the standard FED AVR C compiler. To use it as a standard compiler then turn off the application designer using the *Project* | *Use Application Designer* menu option. It may be turned back on at will.

Once turned off the there will be no updates to any of the files included in the project. However BE WARNED that if the designer is turned back on and the application generated again then all changes to the _Auto.h, _Main.c, and _Lib.c files will be lost.

FED recommend that the C Compiler reference should be read in association with this manual.